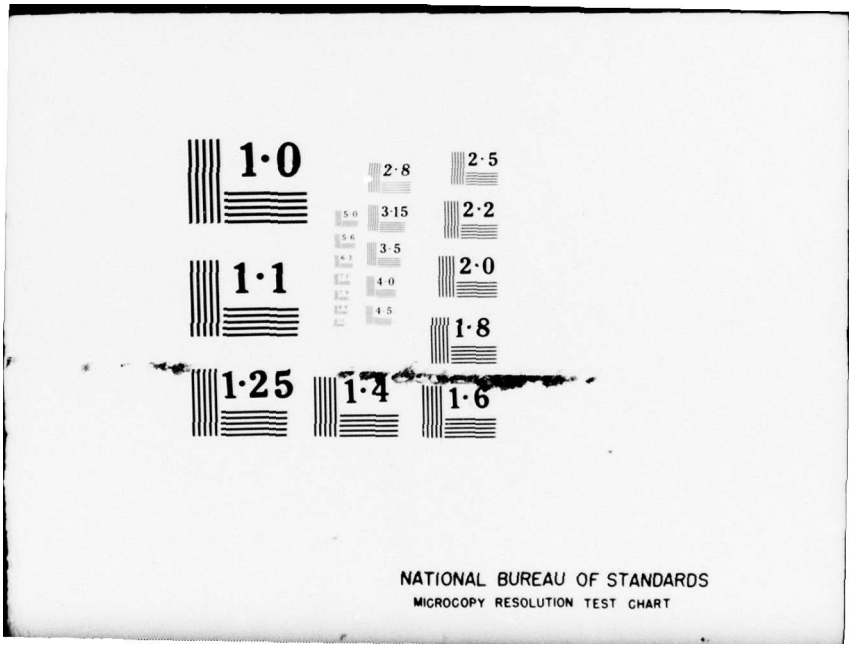
AD-A043 349 AERONAUTICAL RESEARCH LABS MELBOURNE (AUSTRALIA) F/G 9/2
THE SIMULATION LANGUAGE CSMP-10(ARL).(U)
MAY 76 N E GILBERT, P G NANKIVELL
UNCLASSIFIED ARL/AERO. 362 NL

| OF |
ADA
043349

END
DATE
FILMED
9-77
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
6·3
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

ARL/Aero. Note 362

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANIZATION

AERONAUTICAL RESEARCH LABORATORIES

MELBOURNE VICTORIA

Aerodynamics Note 362

THE SIMULATION LANGUAGE CSMP-10(ARL)

N. E. GILBERT and P. G. NANKIVELL

MAY 1976

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANIZATION

AERONAUTICAL RESEARCH LABORATORIES

(14) ARL/Aero. 362

(9) AERODYNAMICS NOTE 362

(6) THE SIMULATION LANGUAGE CSMP-10(ARL)

by

(10) N. E. GILBERT and P. G. NANKIVELL

(11) May 76 (12) 64p.

SUMMARY

A description is given of the 'block oriented' simulation language CSMP-10(ARL), which has been developed from CSMP-10 and is written mainly in FORTRAN IV for a PDP-10 computer. The major improvement made has been to incorporate 'user-defined' blocks, which are written as FORTRAN subroutines. A large number of outputs may be defined within these subroutines by using 'dummy' or 'user output' blocks. Two other major improvements have been made, both enabling the saving of appreciable core storage. Firstly, arrays necessary to store information on each block used are automatically expanded to the size determined by the user. Secondly, the language is divided into a modelling program, which is used to perform the model simulation and store the output in a binary file on a specified storage device, and an output program, which is used to print and plot the character conversion of the binary file.

POSTAL ADDRESS: Chief Superintendent, Aeronautical Research Laboratories,
Box 4331, P.O., Melbourne, Victoria, 3001, Australia.

CONTENTS

1. INTRODUCTION

Programming of both continuous and discrete dynamic systems for digital computers has been aided considerably in recent years by the development of simulation languages. These languages enable the user to make a relatively simple transition from the environment of an analogue computer to that of a digital computer. Simulation languages have advantages over high level programming languages such as FORTRAN and ALGOL of economy in programming time through the sorting of statements and ease of modification, use of centralised integration, and specialised output facilities. There are two types of simulation language, 'block oriented' and 'equation oriented': the latter resemble a high level programming language such as FORTRAN, while the former are expressed in coded form with the aid of a block diagram.* The interactively controlled 'block oriented' language CSMP-10[1] became available for use on the PDP-10 computer at ARL but, because of its inability to handle complex algebraic expressions, was of limited usefulness. Consequently, a modified version, called CSMP-10(ARL), has been developed whose major improvement is the incorporation of user-defined blocks which are written as FORTRAN subroutines. CSMP-10 was developed from a series of languages, namely (in order of development) PACTOLUS[2] for an IBM-1620, 1130-CSMP[3,4] for an IBM-1130, and CSMP-9[5] for a PDP-9. Besides CSMP-10(ARL), there have been a number of other off-shoots with extended capabilities, including 1130 CSMP[6,7] written for an IBM-1130, and allowing the use of five short user-defined FORTRAN subroutines and several other improvements. Another version, ***CSMP,[8] offers several improvements which also appear in CSMP-10(ARL), but allows only two short user-defined subroutines written in BASIC.

An outline description of CSMP-10(ARL) is given in Section 2, which includes also the improvements made to CSMP-10. Section 3 shows how a specific mathematical model may be represented using various statements, while Section 4 gives the Teletype responses necessary for the beginner to complete the simulation of his specified mathematical model. Details of more advanced features of the language are given in Section 5 to 9. In addition to the many changes made to CSMP-10, the Teletype messages and corresponding responses have been completely revised, and much greater detail is provided here than in descriptions of previous versions of the language.

The language CSMP-10(ARL) is currently being used at ARL in the computer simulation of the behaviour of helicopters while under manual or automatic flight control.

---

* A block diagram (e.g. Fig. 2) consists of a number of linked modules (called blocks), each one representing a particular function or operation.

2. OUTLINE DESCRIPTION OF CSMP-10(ARL)

2.1 General

CSMP-10(ARL) is written mainly in FORTRAN IV for the PDP-10 FORTRAN compiler 'F40 Version-27' running under the operating system 'FOROTS'. The language consists of a modelling program named BOMMP (Block Oriented Mathematical Modelling Program) and an output program named TRANS (Translation). In conjunction with various statements, control parameters, and FORTRAN subroutines defined by the user in order to represent a specific mathematical model, the modelling program is used to perform the simulation and store the output in a file in binary form on a specified storage device. The output program is then used to print and plot the character conversion of the binary file in either tabular or graphical form. The output program may be used independently of the modelling program as a general purpose output program (provided data are supplied in the appropriate format). Only a brief description of its use is therefore included here (in Section 4.2); a complete description is given in Reference 9. Although the modelling program is designed to be run interactively from a Teletype, it may also be run non-interactively during Batch or On-Line processing. In the modelling program, the subroutines CPU, EXPAND, FCHECK, and TTYCHK are written in MACRO-10, the assembly language for the PDP-10.

The modelling program allows a specific mathematical model of a dynamic system to be represented by control parameters and three types of statements, viz, configuration, parameter, and function statements. The configuration statements describe the block structure of a particular model; each statement consists of a block number, a block type (e.g. integrator, adder), and a list of which other blocks (up to three) supply the block with input signals. The parameter statements specify numerical values of parameters associated with the configuration statements, while function statements specify coordinate pairs used to generate a function. The configuration statements may be listed in any order; the modelling program (BOMMP) sorts them into an executable order. Following these statements, integration and output control parameters are specified. Integration is performed numerically using a second order Runge Kutta (i.e. Modified Euler) method with a fixed integration interval. The user specifies the lower and upper time limits and the interval thus giving equispaced time steps at which the calculations are performed and required output values are stored. For the integration method used (see Section 6.2), there are two stages of calculation. The second stage provides a solution of the simulation at the present time step, while the first stage provides a solution at a time value midway between the present and previous time steps. This 'mid-point' time value is referred to as a 'half time value' or 'half time step'.

2.2 Improvements to CSMP-10

(*a*) *User-defined blocks*

Because of its inability to handle complex algebraic expressions either directly in the configuration statements or indirectly in user-defined subroutines written in FORTRAN, CSMP-10 cannot be considered a very useful general purpose simulation language. The major improvement made in CSMP-10(ARL) therefore has been to incorporate user-defined blocks, which are written as FORTRAN subroutines and are compiled and loaded into core with the compiled version of BOMMP. The current version of BOMMP allows a maximum of fifteen of these blocks; however, this number may be increased indefinitely within the limits of the computer core storage by simple coding alterations to BOMMP. These blocks may have only three sorted block inputs (as declared in the configuration statement), but may have any number of unsorted block inputs. 'Dummy' or 'user output' blocks have been created whose output value is defined in the appropriate user-defined subroutine, thus allowing a large number of additional outputs for each of these blocks.

(*b*) *Expandable arrays*

With CSMP-10, the total number of blocks is limited to seventy-five, which includes a maximum of twenty-five integrator and unit delay blocks, and three function blocks. These limits could be increased by resetting the appropriate DIMENSION statements in BOMMP, but to do so would reserve a large amount of core storage that may not be required for most models. Therefore, the facility has been introduced in CSMP-10(ARL) whereby the user specifies

(i) the maximum block number,
(ii) the total number of integrator and first order lag blocks,
(iii) the number of unit delay blocks, and
(iv) the number of function blocks.

The appropriate arrays in BOMMP are then automatically expanded to the necessary size so that a minimum of core storage is used. The provision of this facility required the inclusion of a subroutine named EXPAND, which as previously stated is written in the assembly language MACRO-10 for the PDP-10. For other computers, this subroutine would have to be rewritten in the appropriate language; alternatively, the facility may be removed and array limits set as in CSMP-10.

(*c*) *Block output*

In CSMP-10, a significant amount of core storage is used to code the output section of the program. By dividing the language into a modelling program and an output program, the core storage required for coding is significantly reduced. The size of the problem that can be handled in CSMP-10(ARL) is then determined by the requirements of the modelling program alone. A large number of improvements to the output part of the language have been effected without reducing the size of the problem that can be handled. Also, because the 'raw' output data are stored in a file in binary form by the modelling program, various forms of output of the same data may be obtained without repeating the simulation. As a further saving, an output value is only stored if it changes by more than a specified percentage from the last value stored at a previous time step.

(*d*) *Other Improvements to CSMP-10*

In addition to the three major improvements described above, a large number of minor improvements have been made, many of them being corrections to errors in the available version of CSMP-10. The significant improvements include:

(i) an improved iterative method of solving an implicit equation;
(ii) improved integration facilities which include a first order lag block and 'hold' and 'reset' operations for the integrator block;
(iii) a function block with any number of arbitrarily spaced abscissae and where, in addition to linear interpolation, either a polynomial of degree two or higher, or a user-defined function, may be used to calculate the function value;
(iv) a debugging facility;
(v) alphanumeric labelling of each block; and
(vi) the capability of continuing execution of the model at a later stage.

3. MODEL REPRESENTATION

To represent a dynamic system in CSMP-10(ARL), the user should first produce a block diagram of the type suitable for programming an analogue computer. The list of available block functions in CSMP-10(ARL) is given in Table 1 (see Section 3.1 for associated notation). More complex mathematical operations may be defined by the user as user-defined blocks written as FORTRAN subroutines. The block diagram should then be translated into configuration, parameter, and function statements. These statements, which are outlined briefly in Section 2.1, are described in detail below in Section 3.1. To complete the model representation, the user needs to specify integration and output control parameters (see Sections 2.1 and 4.1.6).

In order to understand many of the features of the language, particularly the more advanced ones, it is important to know the algorithm for sorting the configuration statements into an executable order and the processes by which the model is executed. These two processes are therefore described below in Sections 3.2 and 3.3 respectively.

3.1 Types of Statements

As outlined in Section 2.1, there are three types of statements associated with the model representation, viz, configuration, parameter, and function statements. The notation associated with these statements and the integration parameters is defined as follows (see Fig. 1 for corresponding block diagram notation):

B	block number (i.e. an integer identifier)
B1, B2, B3	blocks which supply input signals to a given block (referred to as inputs)
M	integer parameter used only with the block types F, I, T, T1, V, and Y; zero for all other blocks (see Table 1 and appropriate sections)
P1, P2, P3	parameters associated with a given block
T	block type (library given in Table 1; each type has an associated reference number)
X	output value of block B
X1, X2, X3	output value of blocks B1, B2, and B3 respectively
h	time interval for integration
t	time
t_0	initial time

Errors and their diagnostic messages relating to the above specification statements are presented in Appendix B.

TABLE 1

Block Type Library

(refer Appendix A)

Name	Type	Type No.	No. of Inputs	No of Parameters	Description
Bang bang	B	2	1	0	X = 1 if X1 $\geqslant$ 0 = −1 if X1 < 0
Dead space	D	4	1	2	X = X1−P1 if X1 > P1 = X1−P2 if X1 < P2 = 0 if P2 $\leqslant$ X1 $\leqslant$ P1
Function	F	6	1	1	X = f(X1). Interpolation from a function generated by co-ordinates with arbitrarily spaced abscissae; linear if P1 = 0 or 1, of degree P1 if P1 > 1, user-defined if P1 = −1, −2, or −3. Uses P3 and M internally (see Section 3.1.3)
Gain	G	7	1	1	X = P1*X1
Half power	H	8	1	0	X = $\sqrt{X1}$
Integrator	I	9	3	3	X = P1 + $\int_{t_0}^{t}$ y dt where y = X1(1+P2)+P3; if X2 ≠0; RESET TO P1 if X3 ≠ 0. Uses M internally (see Section 6.2).
Jitter	J	10	0	0	X = {random number uniformly distributed between −1 and +1}
Constant	K	11	0	1	X = P1

TABLE 1—continued

Name	Type	Type No.	No. of Inputs	No. of Parameters	Description
Limiter	L	12	1	2	X = P1 if X1 > P1 = P2 if X1 < P2 = X1 if P2 ⩽ X1 ⩽ P1
Magnitude	M	13	1	0	X = |X|
Negative clipper	N	14	1	0	X = X1 if X1 > 0 = 0 if X1 ⩽ 0
Offset	O	15	1	1	X = X1+P1
Positive clipper	P	16	1	0	X = X1 if X1 < 0 = 0 if X1 ⩾ 0
Quit	Q	17	2	0	Terminates run if X1 > X2; types message *RUN TERMINATED BY A Q BLOCK*
Relay	R	18	3	0	X = X2 if X1 ⩾ 0 = X3 if X1 < 0
Time pulse generator	T	20	1	1	Generates pulse train of period P1 starting when X1 ⩾ 0. Uses P2 and M internally (see Appendix A; also, Section 8.2 for use in discrete system simulation)
First order lag	T1	30	3	2	Solution of the first order differential equation $X+P2\,dX/dt = X1+X2+X3$ where X = P1 when $t = t_0$. Uses M internally (see Section 6.3)
Unit delay	U	21	1	1	X = P1 when $t = t_0$ = X1 at $t-h/2$ when $t > t_0$. Uses P2 internally when the input B1 is a U block (see Section 6.1; also Section 8.2 for use in discrete system simulation)
User output	UO	3	1	0	X is defined in the user-defined subroutine associated with the input B1 (see Section 7)

TABLE 1—continued

Name	Type	Type No.	No. of Inputs	No. of Parameters	Description
User-defined	Um	30+m	3	3	X is defined by the user in subroutine USERm, where m is an integer, 1 to 15 (see Section 7)
Vacuous	V	22	0	1	Use with wye block to solve an implicit equation (e.g. $y = f(y)$). P1 is the initial approximation of y, i.e. y_0. Uses P2, P3, and M internally (see Section 8.1)
Weighted summer	W	23	3	3	X = P1*X1 + P2*X2 + P3*X3
Multiplier	X	24	2	0	X = X1*X2
Wye	Y	25	2	2	Used with vacuous block to solve an implicit equation (e.g. $y = f(y)$). X = y; P1 is the error tolerance for convergence, Ec; P2 is the maximum number of iterations, Ni; if not set, Ni is assumed to be 20. Uses P3 and M internally; also, dual use of P2 (see Section 8.1)
Zero order hold	Z	26	2	1	X = P1 when $t = t_0$ = X1 if X2 > 0 when $t > t_0$; X is unchanged if X2 ≤ 0 when $t > t_0$; P2 is used internally to store X at the previous half time step (see Section 8.2 for use in discrete system simulation)
Summer	+	27	3	0	X = ±X1 ± X2 ± X3. The inputs B1, B2, B3 may be preceded by a plus or a minus sign in the configuration statement
Divider	/	29	2	0	X = X1/X2
Inverter	—	28	1	0	X = −X1
Blank		5	0	0	Causes the previous statement with the same block number to be deleted but not replaced

3.1.1 Configuration Statements

The configuration statements describe the block structure of a model and have the form

B, T, B1, B2, B3; "label" $ "comment"

The arguments are separated by commas (as shown) or blank characters. The modelling program reserves block number 1, whose output value is the current time, t. A blank block number (i.e. two consecutive 'carriage-returns') terminates the set of configuration statements. The library of available block types is given in Table 1. The following examples of configuration statements illustrate the various ways these statements may be typed (explanations given below):†

2 D 16 : ALPHA $ IN RADIANS

3, I, 4, 5, 6 $ACCELERATION

7, K; PHI

8 W 12 9 B3 is blank so that X3 is assumed zero

The inputs B1, B2, and B3 must be valid block numbers, or zero or blank to signify the absence of an input. If the block type is a 'summer' (i.e. T is '+'), the inputs B1, B2, and B3 may be negative integers. If, for example, B1 equals −3, this means that the output value of block number 3 will be multiplied by −1 when used as the input signal to the given 'summer' block. Although the general definition of a block type may assume the presence of specific inputs, one or more of these inputs may be left blank, in which case the corresponding value of each of these inputs is assumed to be zero (e.g. in the above examples, B3 for block number 8). Block types that often do not require all their specifiable inputs are 'integrator' (I), 'first order lag' (T1), 'user-defined' (Um), 'weighted summer' (W), and 'summer' (+).

The semi-colon in the configuration statement is optional; when used, the ten characters following may be used to provide an alphanumeric label for B, which is retained when storing the model (see Section 4.1.5) and is used to label the printed and plotted output. The dollar sign terminator is also optional; all characters following it are ignored and may therefore be used as comments.

The configuration statements may be input in any order. The sort process, described in Section 3.2, determines the order in which the statements are executed. If the user types a configuration statement having a block number, B, that has already been used as the block number of a previous statement, the previously defined statement is replaced by the new statement unless the type, T, of the new statement is blank, in which case the previous statement is deleted but not replaced (see examples in Section 4.1.4).

3.1.2 Parameter Statements

Parameter statements specify parameter values, e.g. initial conditions and multiplication constants, associated with the configuration statements (see Table 1), and have the form

B, P1, P2, P3 $ "comment"

As with configuration statements, the arguments may be separated by commas or blanks, and two consecutive 'carriage-returns' terminate the set of parameter statements. Here again, the dollar sign followed by a comment is optional, and previous parameter statements may be replaced by repeating the block number. The parameters P1, P2, and P3 are associated with the

† Teletype messages typed by the user are shown in bold upper case; messages typed by the appropriate computer program are shown in italic upper case. Comments on Teletype messages are shown in upper and lower case print alongside the appropriate message.

block number B appearing in the configuration statements. For example, when the parameter statement

21, 1.5 $ INITIAL CONDITION FOR Y

is typed for a model with the previously defined configuration statement

21, I, 103

the initial value of block number 21 is set equal to 1.5 (as defined by the specification for an integrator block type in Table 1).

Aecause storage for the parameters is allocated for each of the blocks defined in the configuration statements, any parameters not set will assume a value of zero. To delete the parameters of a particular block, the user sets the parameters equal to zero by typing only the block number.

3.1.3 Function Statements

Function statements specify coordinate pairs used to generate a function for each block of type 'F' defined in the configuration statements. The appropriate block number is first typed following the Teletype message

BLK NO. =

and each coordinate pair, i.e. abscissa followed by ordinate, is then typed on a separate line following the message

COORD PAIRS:

(see example in Section 4.1.4). Two consecutive 'carriage-returns' terminate the list of coordinate pairs. The first of the two Teletype messages above is then repeated so that further functions may be specified: a blank block number (i.e. 'carriage-return') terminates the set of function statements.

The coordinate pairs define a function with arbitrarily spaced abscissae. Between each adjacent pair of abscissae, the value of the function is obtained by interpolation. The interpolation is linear if P1 = 0 or 1, of degree P1 if P1 > 0, and user-defined if P1 = −1, −2, or −3 (see Appendix C). Outside the abscissae limits, the function value is obtained by extrapolation. The coordinate pairs may be typed in any order; on typing each pair, the appropriate list is rearranged such that the abscissae are in order of increasing magnitude. While entering the list, coordinate pairs may be deleted or replaced (see example in Section 4.1.4). The list may also be modified at subsequent stages (see Section 5.3). The internal use of the parameter P3 and integer parameter M in locating the position of the data for each function (F) block in the appropriate expandable array (used to store all the function coordinate pairs) is described in Appendix C.

For a polynomial of degree P1, at least P1 + 1 coordinate pairs must be defined. There is no limit, subject to sufficient core storage being available, to the number of coordinate pairs that may be defined for each function. However, the number, n say, of F blocks specified when dimensioning the appropriate expandable array (see Section 4.1.2) must not be less than the actual number of F blocks used, and the total number of coordinate pairs of all F blocks must not exceed $15n$ (i.e. to exceed an average of 15 points per F block, n must be made larger than the actual number of F blocks used).

3.2 Sort Process

The purpose of the sort process is to arrange the blocks, represented by their configuration statements, into an order such that for each statement the input signals have been previously defined. The list of blocks obtained in this way is termed the 'sort list', and execution of the blocks in this list for each half time step is referred to as 'execution of the sort list'. User output (UO) blocks are excluded from the sort list because their output is calculated when the associated user-defined block is executed. However, when storing the configuration statements (see Section

4.1.5), statements with a UO block type are stored immediately following the associated statement with a user-defined block type.

The algorithm for determining the sort list is described as follows:

(1) Enter into the sort list the constant (K) blocks in ascending order of block number followed by the unit delay (U) blocks, also in ascending order of block number.
(2) Select the lowest numbered unsorted block and then go to (4).
(3) Select the next lowest unsorted block number.
(4) If the block numbers of all the inputs (i.e. B1, B2, and B3) have (a) already been entered into the sort list or (b) are themselves defined as integrator (I) or first order lag (T1) blocks,† enter this block into the sort list and then go to (2); otherwise go to (3).

3.3 Model Execution Process

Initially, the output value of each block is set equal to the value of its first parameter, i.e. X = P1. This has the effect of setting initial conditions for integrator (I), first order lag (T1), unit delay (U), vacuous (V), and zero order hold (Z) blocks, and setting constant (K) blocks to their constant value. The output values of other blocks arbitrarily set in this way are overwritten on initially executing the sort list. The only difference from subsequent sort list executions is that the output values of the U, V, and Z blocks are not set since their output values have already been set equal to P1. At each subsequent half time step, the output values of I and T1 blocks are first calculated (see Section 6.2) and the sort list is then executed. Though calculated before execution of the sort list, the output values of I and T1 blocks are not set until immediately after the output values of the U blocks are set. Because the output value of the K block is set initially and does not change, K blocks are skipped when the sort list is executed.

4. BASIC OPERATIONS OF SIMULATION LANGUAGE

This section explains only the basic operations necessary for a user to be able to initially use the simulation language. Further operations, which allow a much greater degree of flexibility, are described in the next section. The modelling program BOMMP is first used to perform the simulation, and the output program TRANS is then used to obtain printed and plotted output. Both programs are controlled by commands, which are typed following an '*' from the Teletype. The appearance of an '*' signifies the command mode. These commands may be shortened to three characters and are listed with brief descriptions in Tables 2*a* and 2*b*. They are described in much greater detail in Sections 4.1, 4.2, and 5. For the modelling program only, a command string may be formed by a list of commands separated by semi-colons.

The operating procedures are best illustrated by an example. For convenience, the non-linear spring problem of the CSMP-9 and 1130-CSMP manuals is repeated. The problem may be expressed in terms of the differential equation

$$a\ddot{y} + b\dot{y} + f(y) = 0 \quad \text{or} \quad \ddot{y} = -[b\dot{y} + f(y)]/a$$

where $f(y)$ is a non-linear function defined by linear interpolation of a set of coordinate pairs (y, f). A complete description of the problem to be solved is given in Figure 2, which also includes a block diagram representation and the necessary configuration, parameter, and function statements.

4.1 Running the Modelling Program

The running of the modelling program may be conveniently divided into the following seven stages:

(1) Appropriate relocatable binary files are loaded into the computer core storage, and a core image of the resulting modelling program is saved.

---

† The output value of integrator and first order lag blocks are known before the sort list is entered (see Section 6.2).

TABLE 2A

Modelling Program Commands

All commands, except the 'channel' command, may be shortened to three characters

Command	Effect	Section(s) for Reference
CONFIGURATION	Reads in configuration statements	3.1.1, 4.1.4
DEBUG	Provides a complete block description during execution	5.5.3
EXIT	Returns control to the monitor	4.1.7
FILE	Gives the status of the model input, block output and model output channels	4.1.3
FUNCTION	Reads in function statements	3.1.3, 4.1.4
GOE	Initiates execution of the model	4.1.7
INTEGRATION	Reads in integration control parameters	4.1.6
LOOK	Examines block output at final time value	5.5.2
MANUAL	During non-interactive On-Line processing, returns the program to interactive running and to the command mode	5.4
OUTPUT	Reads in output control parameters	4.1.6
PARAMETER	Reads in parameter statements	3.1.2, 4.1.4
PROGRAM	Executes another specified program, e.g. 'TRANS'	5.5.4
RETAIN	Enables model execution to continue at a later stage	5.5.5
RUN	When setting up a model, initiates the command string 'TITLE;CON;PAR;FUN;INT;OUT; GOE'; if the Teletype is not used as the model input channel, the 'TITLE' command is excluded; where part of the string has already been initiated by individual commands, the remaining ones are initiated	5.5.1
STORE	Stores model specification statements in a file	4.1.5, 5.1.1
TITLE	Reads in title	4.1.4
'channel'	Used to change the status of the input and output channels	5.1

(2) Execution of the program is initiated, and the amount of additional storage is determined by specifying the maximum block number and number of special blocks so that the appropriate array dimensions may be expanded to the minimum size necessary. The maximum block number may exceed the total number of blocks used since not all intermediate numbers need be used as block identifiers.

(3) The 'status' of the input and output channels may be examined if required (see Sections 4.1.3 and 5.1 for explanation).

(4) The model is set up by specifying in turn: a title, configuration statements, parameter statements, and function statements (if any).

(5) The model may be stored for future use or reference.
(6) Integration and output control parameters are specified.
(7) The model is run, and on completion, control may be returned to the monitor, or modifications may be made to the model by repeating the appropriate commands so that the model may be re-run.

Stages (3) and (5) are optional, but Stages (1), (2), (4), and (6) must be completed in order before the model is run in Stage (7). Stage (3) may be included anywhere following Stage(2) while Stage (5) must be preceded, though not necessarily immediately, by Stage (4). Stages (3) to (7) are controlled by appropriate commands (see Table 2*a*). Though not shown here, commands may be repeated at any stage. This enables alterations or additions to the model.

For the non-linear spring problem, the operating procedures are described below for each of the above stages.

4.1.1 Stage 1—Loading the Program

Appropriate relocatable binary files, which include the files BOMMP.REL, DUSER.REL, and any user-defined files, are loaded into the computer core using the normal PDP-10 loading procedure, and a core image of the resulting modelling program is saved. The user-defined files contain user-defined subroutines, as well as other functions and subroutines required by the user-defined subroutines, and the file BOMMP.REL is the binary version of the basic modelling program. Because the basic modelling program contains 'CALL' FORTRAN statements for each of the user-defined subroutines USER1, USER2, ... USER15 and the subroutines INTRP1, (NTRP2, INTRP3 used for user-defined interpolation in the function block (see Appendix C), corresponding versions of these subroutines are required when loading the program. 'Dummy' versions, which perform no calculations, are therefore stored in a file whose binary version is called DUSER.REL. By loading DUSER.REL last and using the 'L' switch (i.e. the user types **DUSER/L**), these 'dummy' versions are loaded only if corresponding versions specified by the user have not already been loaded. For the non-linear spring problem, which has no user-defined subroutines, the files BOMMP.REL and DUSER.REL are loaded, and a core image of the program is saved as follows:

```
.R LOADER
*BOMMP                  Basic modelling program
*DUSER/L                'Dummy' versions of user-defined subroutines—loaded last
*$                      'Altmode' character

BOMMP 12K CORE, 194 WORDS FREE
LOADER USED 14+4K CORE

EXIT

.SAVE BOMMP             Produces core image in file BOMMP.SAV; the name need not
BOMMP SAVED               be BOMMP
```

4.1.2 Stage 2—Execution Initiation and Dimensioning of Expandable Arrays

The execution of the program is initiated and the expandable array dimensions are set as follows:

```
.RU BOMMP

MAX BLK NO. = 48

NO. OF: I & TI BLKS, U BLKS, F BLKS = 2, 0, 1

*
```

4.1.3 Stage 3—Status of Input and Output Channels

The modelling program uses three channels for input and output. They are:

(1) an input channel for reading in the model specification statements, viz, title, configuration statements, parameter statements, and function statements;

(2) a model output channel for storing these statements for future use;

(3) an output channel for storing the block output (i.e. output value of each of the blocks) in binary form on a specified storage device during execution.

The 'FILE' command gives the status of each of these channels and may be used at any stage of execution. Initially, the channels are set up as follows, but they may be changed at any stage by using the 'channel' command (see Section 5.1):

```
*FIL

MODEL  I/P FROM   TTY:
BLOCK  O/P      TO LOG1:OUTPT.DAT
MODEL  O/P      TO LOG3:MODEL.MOD

*
```

The number of characters used for all file names (without extension) must not exceed five.

4.1.4 Stage 4—Setting up the Model

The model is set up by specifying in turn (*a*) a title, (*b*) configuration statements, (*c*) parameter statements, and (*d*) function statements (if any). The appropriate commands 'TITLE', 'CONFIGURATION', 'PARAMETER', and 'FUNCTION' control the input of these statements by the modelling program; the latter three are hereafter referred to by their shortened form (i.e. 'CON', 'PAR', and 'FUN'). Each of the three sets of statements is terminated by a blank block number (i.e. an additional 'carriage-return'). The model for the non-linear spring problem may then be set up as shown below, where typical errors are included to illustrate their correction procedure:

```
*TIT

TITLE (LIMIT 60 CHRS)
NON-LINEAR SPRING WITH DASHPOT

*CON

CONFIGURATIONS:

BLK, TYPE, B1, B2, B3
4, /, 17, 6; Y DBLE DOT
5, K                          Typed in error
6, K
5
PREV BLK   5 DELETED          Blank type causes deletion of statement defined previously
9, I, 48;    Y                  for block number 5
17, W, 48, 10
48, I, 4;   Y DOT
10, F, 9

                              Blank block number terminates configuration statements
*PAR
```

```
PARAMETERS:

BLK, P1, P2, P3
9, -10.0
17, 2                          Second parameter omitted in error
17, 2, 1                       Repeated block number causes replacement of parameters
6, -5                            defined previously for block number 17
                               Blank block number terminates parameter statements
*FUN

FUNCTIONS:

BLK NO. = 10

COORD PAIRS:                   Coordinate pairs are read in using the format statement
                                   FORMAT (2E)
-10, -100
-8, -64
-6, -36                        Typed in error
-6, -36                        Repeated abscissa causes replacement
COORD PAIR (-6.0000E+00,  3.6000E+01) DELETED
-4, -16
0, 0
-2, -4                         Coordinate pairs may be in any order
2, 4
8, 64
5, 30                          Typed in error
5                              Blank ordinate causes deletion without replacement
COORD PAIR ( 5.0000E+00,  3.0000E+01) DELETED
4, 16
10, 100
6, 36
                               Blank abscissa terminates coordinate pairs
BLK NO. =                      Blank block number terminates function statements

MODEL COMPLETE

*
```

The message

MODEL COMPLETE

(see above example) is typed by the program only when coordinate pairs for all the function blocks have been specified. If there are no function blocks, the message is typed following the completion of the parameter statements. Even if there are no parameters to be specified, the 'PAR' command must still be used following the 'CON' command.

4.1.5 Stage 5—Storing the Model

By using the 'STORE' command, the current model specification statements may be stored for future use in a file which is referred to as the model output channel. For the present example, the complete model is stored in 'LOG3:MODEL.MOD' (refer to initial status of the model

output channel in section 4.1.3), but each set of statements may be stored in separate model output channels; however, the model output channel must be redefined prior to each set being stored (see Section 5.1.1). The title is included in the set of configuration statements. Comments following the optional statement terminator '$' are ignored and hence are not stored (see Section 5.3 if comments are required to be stored). The appropriate Teletype messages and responses here are:

***STO**

*CON, PAR, FUN, OR ALL :***ALL**

*

The configuration, parameter, and function statements are each stored in ascending order of block number. However, by typing **CS** in place of **CON**, and **AS** in place of **ALL**, the configuration statements are stored in sorted order of block number.

Parameter statements that have a zero value for P1, P2, and P3 are not stored.

4.1.6 Stage 6—Specification of Integration and Output Control Parameters

The commands 'INTEGRATION' and 'OUTPUT', which are hereafter referred to by their shortened form (i.e. 'INT' and 'OUT'), allow the integration and output control parameters to be set; the order of these commands is optional. For the integration, which is performed using the second order Runge Kutta (i.e. Modified Euler) method, the lower and upper time limits and the interval are specified (each in the same unit). For the output control, the required block numbers are first listed. They may be typed on more than one line, each line consisting of a maximum of 72 characters; two consecutive 'carriage-returns' terminate the list. If all the block numbers are required, the user may type **ALL** (which may be shortened to **A**) in place of the individual block numbers, in which case a single 'carriage-return' terminates the list: the block output values are stored in ascending order of block number. By typing **AS** in place of **ALL** or **A**, the block output values of all the blocks are stored in sorted order of block number. An output value is only stored if it changes by more than a prescribed percentage from the previously stored value; this percentage is specified, followed by the interval of time between output values in the same unit as the integration parameters. The appropriate Teletype messages and responses for the present example are:

***INT**

INTEGN PARAMS; LOWER, UPPER, INTERVAL = **0, 20, 0.1**

***OUT**

O/P BLKS
9, 4, 48, 10 Two consecutive 'carriage-returns' terminate the list

O/P PARAMS; % CHANGE REQRD, INTERVAL = **.01, 1**

*

If the output interval is less than the integration interval, the output values are recorded at every half time step of the integration (see Sections 2.1 and 6.2). To execute the model only at the initial time value, the lower and upper integration limits should both be set equal to the initial time value; the integration interval need not then be set.

On using the 'OUTPUT' command, a 'TITLE' command is first implied if a title has not already been specified (i.e. the program requires the user to type a title).

4.1.7 Stage 7—Executing the Model

The 'GOE' command initiates execution of the complete model. Upon completion, the run CPU time is given and control is returned to the command mode. If required, alterations may then be made to the model before re-execution (see Section 5.3). Eventually, control is returned to the monitor by the 'EXIT' command. Use of the 'GOE' and 'EXIT' commands is illustrated as follows:

```
*GOE

** RUNNING **

RUN CPU TIME:  1.25 SEC.

*EXI

END OF EXECUTION
CPU TIME: 5.74 ELAPSED TIME: 13:58.74
EXIT
```

The model execution may be halted before completion by typing ↑, in which case execution is completed up to the next time value for which the block values specified for output are required. The message

RUN TERMINATED BY A "↑"

is typed by the program and control is returned to the command mode.

4.2 Running the Output Program

The output program TRANS is controlled by commands in exactly the same way as the modelling program. These commands are listed with brief descriptions in Table 2*b*. The program provides Teletype and line printer output in either tabular or graphical form and incremental plotter output in the form of 'strip' plots (of the form produced by a multi-channel chart recorder) and 'overlay' plots (a single graph for up to eight output variables). The block output file obtained by running BOMMP, which then becomes the input file for TRANS, includes the time limits and interval, a title, and labels for each output variable. Subject to some restrictions, these may be redefined. For graphical output, convenient scales are calculated by TRANS such that each curve is fully shown; however, these scales may also be redefined by the user.

Only the commands necessary to produce Teletype tabular and graphical output (i.e. 'PRCOLUMN' and 'PRPLOT') are described here; a complete description of all the features of TRANS is given in Reference 9.

4.2.1 Setting up the Program

The user initiates execution of the program by first running the core image version TRANS.SAV of the relocatable binary file TRANS.REL, and then specifying the filename (*without extension*) of the block output file (OUTPT for the non-linear spring problem) produced by running BOMMP. The output program then types the title, date and time of creation of the block output file, integration interval, run CPU time, and time parameters, followed by an '*', which signifies the command mode. An example of how to set up the program in the above manner is shown below.

TABLE 2B

Output Program Commands

All commands may be shortened to three characters

Command	Effect	Section(s) for Reference
EXIT	Returns control to the monitor	4.2.1
GOE	Obtains the tabular or graphical output for the commands 'PRCOLUMN', 'PRPLOT', 'PLSTRIP', and 'PLOVERLAY'	4.2.1
LABEL	Redefines labelling information	—*
PLOVERLAY	Specifies 'overlay' plotting on the incremental plotter	—*
PLSTRIP	Specifies 'strip' plotting on the incremental plotter	—*
PRCOLUMN	Specifies tabular output on the disk or Teletype in the form of printed columns	4.2.2
PRPLOT	Specifies graphical output on the disk or Teletype	4.2.2
RUN	Equivalent to the commands (in order) 'PRCOLUMN', 'PRPLOT', 'PLSTRIP', and 'GOE'; if any form of output is not required, a 'carriage-return' is typed in place of the block numbers	—*
SCALE	Redefines graphical output scales	—*
TIME	Redefines time parameters	—*

* See Reference 9.

.RU TRANS

I/P FILENAME = **OUTPT**

NON-LINEAR SPRING WITH DASHPOT

I/P FILE RECORDED ON 21-APR-76 AT 16:57

INTEGN INT = 1.0000E-01; RUN CPU TIME = 1.25 SEC.

TIME FROM 0.0000E−01 TO 2.0000E+01 IN STEPS OF 1.0000E+00

*

Although the filename extension is not specified, the extension name 'DAT' is assumed. If, in the above example, a file named OUTPT.DAT is not found on the disk, the message

OUTPT.DAT NOT ON DSK

I/P FILENAME =

is typed by the program so that a new filename may be specified.

A particular type, or types, of output may be specified by using the appropriate commands listed in Table 2*b*; to complete the output process, the 'GOE' command should follow, after which control is returned to the command mode. Repetition of the appropriate commands for each type of output before the 'GOE' command, deletes the effect of the previous use of the command; repetition after the 'GOE' command, results in additional output. Control is returned to the monitor by the 'EXIT' command.

4.2.2 Teletype Output

The commands 'PRCOLUMN' and 'PRPLOT' are used to obtain tabular and graphical output respectively on either the line printer or Teletype. For each command the required block numbers are first listed. They may be typed on more than one line, each line consisting of a maximum of 72 characters, and the list is terminated by two consecutive 'carriage-returns'. If all the block numbers are required, the user may type **A** in place of the individual block numbers, in which case a single 'carriage-return' terminates the list. When specifying blocks by their individual block numbers, a maximum of a hundred blocks may be processed each time the 'GOE' command is used; however, all blocks are processed when the user types **A** instead, even if there are more than a hundred blocks. Following the Teletype message

IS O/P TO TTY REQRD:

typed by the program, the user types **Y** if the output is to be printed directly on the Teletype, or any other character (including a 'carriage-return') if the output is to be stored in a disk file so that the resulting file can subsequently be printed on the line printer (see Reference 9), This latter form of output contains more information than the Teletype output; blocks are identified by their number rather than by a label on the Teletype output. On completing the Teletype responses to either the 'PRCOLUMN' or 'PRPLOT' commands, the program returns control to the command mode. The 'GOE' command is then used to obtain the Teletype output. It should be noted that both tabular and graphical output cannot be obtained on the Teletype by using a single 'GOE' command following the 'PRCOLUMN' and 'PRPLOT' commands. Convenient plotting scales are calculated by the output program and are shown at the head of each Teletype plot.

The following example shows how both tabular and graphical Teletype output may be obtained for the non-linear spring problem:

***PRC**

PRINTING IN COLUMNS:

BLKS
A List terminated by a single 'carriage-return'

IS O/P TO TTY REQRD: **Y**

***GOE**

*** RUNNING ***

(See Section 1 of Appendix D for Teletype output)

***PRP**

PRINTER PLOTS:

BLKS
9, 4 List terminated by two consecutive 'carriage-returns'

IS O/P TO TTY REQRD: **Y**

***GOE**

*** RUNNING ***

(See Section 2 of Appendix D for Teletype output)

***EXI**

END OF EXECUTION
CPU TIME: 2.27 ELAPSED TIME: 9:9.82
EXIT

5. FURTHER OPERATIONS OF MODELLING PROGRAM

In the previous section, the basic operations necessary to run both the modelling program and output program were described. In this section, further operations of the modelling program are described that allow much greater flexibility. In particular, facilities are described that allow repeated alteration to the model and repeated execution of the model without exiting from the modelling program.

5.1 Changing the Status of Input and Output Channels

The three channels used for input and output are described in Section 4.1.3, where the initial status of each channel is given by the 'FILE' command. If the user wishes to re-run a model, perhaps after making some alterations, it may be desirable to alter the status of one or more of these channels. The logical unit and filename for a particular channel is altered by the program command (termed a 'channel' command)

LOGn: "filename" ← "channel"

where

(1) n is an integer, 1 to 12, representing the logical unit number; the following list gives those that are reserved, LOG5 being permanently reserved:

LOG1—block output channel (initially);
LOG3—model output channel (initially)
LOG5—output file named BOMMP.OUT which contains the Teletype responses typed by the user for subsequently running the modelling program non-interactively (see Section 5.4);
LOG12—output file named DEBUG.OUT which contains a complete block description (i.e. values for each block of X, X1, X2, X3, M, P1, P2, and P3) for each half time step during execution; the file is only opened, and logical unit only reserved, following an initial 'DEBUG' command (see Section 5.5.3) when the output is not required on the Teletype.

(2) "filename" is a filename without an extension and with a maximum of five characters.

(3) "channel", which is one of the following list, represents a particular input or output channel:

I/P—input of model specification statements (to run the model);
MODEL—output of model specification statements (to store the model);
O/P—block output.

The above channel descriptors may be shortened to one character (e.g. 'I/P' may be shortened to 'I').

The Teletype may be used as the model input or model output channel, in which case the 'channel' command takes the form

TTY: ← "channel"

The Teletype is initially used as the model input channel. The model specification statements entered in this way are separate from the commands and control parameters read in using the Teletype. The status of the model output and block output channels remains the same unless changed by a further channel command. However, if the status of the model input channel has been changed from the Teletype to reading from a file, it reverts to the Teletype once input is completed (see Examples 1 and 2 in Section 5.1.2). If further specification statements are required to be read in from another file, the status of the model input channel would have to be changed from the Teletype to the new file.

Whenever the status of a channel is changed, either by a 'channel' command or upon reversion to the Teletype for the model input channel, the status of the appropriate channel is typed on the Teletype by the program as shown in the examples in Sections 5.1.1 to 5.1.3.

The use of different logical units enables files to be read from, or written onto, a specific device other than the disk (i.e. DECtape or magnetic tape). To use this facility, the 'ASSIGN' monitor command is used; for example, to assign DECtape DTA3 to LOG1, the user types **AS DTA3 1** to the monitor prior to running the modelling program.

5.1.1 Model Output

In the non-linear spring problem of Section 4, by using the 'STORE' command, the model specification statements were stored in 'LOG3:MODEL.MOD' (see Sections 4.1.3 and 4.1.5). The 'STORE' command allows any one, as well as all, of the separate groups of specification statements (i.e. configuration, parameter, and function statements) to be stored in the file defined as the model output channel. The three examples below show how (1) all the specification statements may be stored in 'LOG6:SPRING.MOD', (2) the configuration, parameter, and function statements may be stored separately in 'LOG7:SPCON.MOD', 'LOG8:SPDAT.MOD', and 'LOG7:SPFUN.MOD' respectively, and (3) all the specification statements may be listed on the Teletype by using the Teletype as the model output channel (headings are deleted).

Example 1

***LOG6:SPRNG←M**

MODEL O/P TO LOG6:SPRNG.MOD

***STO**

*CON, PAR, FUN, OR ALL :***ALL**

*****

Example 2

***LOG7:SPCON←M**

MODEL O/P TO LOG7:SPCON.MOD

***STO**

*CON, PAR, FUN, OR ALL :***CON**

***LOG8:SPDAT←M**

MODEL O/P TO LOG8:SPDAT.MOD

***STO**

*CON, PAR, FUN, OR ALL :***PAR**

```
*LOG7:SPFUN←M

MODEL O/P   TO  LOG7:SPFUN.MOD

*STO

CON, PAR, FUN, OR ALL :FUN

*
```

Example 3

```
*TTY:←M

MODEL O/P   TO  TTY:

*STO

CON, PAR, FUN, OR ALL :AS
```

Configuration statements in sorted order (see Section 4.1.5)

```
NON-LINEAR SPRING WITH DASHPOT
    6   K
    9   I      48          ;     Y
   10   F       9
   17   W      48   10
    4   /      17    6     ;Y DBLE DOT
   48   I       4          ;    Y DOT

    6  -5.0000E+00
    9  -1.0000E+01
   17   2.0000E+00   1.0000E+00

   10
-1.0000E+01  -1.0000E+02
-8.0000E+00  -6.4000E+01
-6.0000E+00  -3.6000E+01
-4.0000E+00  -1.6000E+01
-2.0000E+00  -4.0000E+00
 0.0000E-01   0.0000E-01
 2.0000E+00   4.0000E+00
 4.0000E+00   1.6000E+01
 6.0000E+00   3.6000E+01
 8.0000E+00   6.4000E+01
 1.0000E+01   1.0000E+02

*
```

Usually, when reading from a file used as the model input channel, all the statements would be read in. However, it is possible that the different types of statements may need to be read in from different sources (i.e. Teletype and various files) without reading in all the information stored in a particular file. For example, two files may each contain a set of configuration, parameter, and function statements. The user may wish to read the configuration statements from one file (File A say), the parameter statements from the Teletype, and the function statements from the other file (File B say). The procedure would be as follows:

(1) Use the 'channel' command so that File A is used as the model input channel.

(2) Type **CON** to read in the configuration statements.

(3) Use the 'channel' command so that the Teletype is used as the model input channel; the parameter and function statements are therefore not read in from File A.
(4) Type **PAR** followed by the parameter statements.
(5) Use the 'channel' command so that File B is used as the model input channel.
(6) Type **FUN** to read in the function statements; the configuration and parameter statements are therefore skipped in File B.

Although the user is free to redefine the data source for each type of statement, it should be remembered that when setting up the model, the commands 'CON', 'PAR', and 'FUN' (if any functions) must be typed in order; the 'TITLE' command would also be required if reading the configuration statements from the Teletype. Until the model is complete, modifications should not generally be attempted by repeating the commands (except for the 'TITLE' command). The model input channel only reverts to the Teletype on reaching the end of a file. If a file is not read to its end therefore, as in the case of File A in the above example, a 'channel' command must be used to either revert to the Teletype or change to another file. On making an error in the above procedures, an appropriate diagnostic error message (Appendix B) is given, and control is returned to the command mode with the model unaffected.

5.1.2 Model Input

Having already stored the specification statements during a previous run of the modelling program, the user may wish to set up the model using these stored statements. Alterations may then be made after all the statements have been read in (see Section 5.3) from the current input file. Because the title is included in the configuration statements when using the 'STORE' command, the 'TITLE' command should not be used unless the title is to be modified (see Section 5.3), in which case it should be used after the configuration statements have been read in. If it is used before, the title included in the configuration statements will overwrite the intended title. Consider Examples 1 and 2 given in Section 5.1.1, where the same specification statements are stored in two different ways. The method of reading these statements for both examples is:

Example 1

```
*LOG2:SPRNG←I

MODEL I/P FROM  LOG2:SPRNG.MOD

*CON

CONFIGURATIONS :

NON-LINEAR SPRING WITH DASHPOT

 BLK  TYPE  B1  B2  B3
   4    /   17   6      ;Y DBLE DOT
   6    K
   9    I   48           ;   Y
  10    F    9
  17    W   48  10
  48    I    4           ;  Y DOT

*PAR

PARAMETERS :

 BLK      P1            P2            P3
   6  -5.0000E+00
   9  -1.0000E+01
  17   2.0000E+00   1.0000E+00
```

***FUN**

FUNCTIONS :
BLK NO. 10

COORD PAIRS :

```
-1.0000E+01  -1.0000E+02
-8.0000E+00  -6.4000E+01
-6.0000E+00  -3.6000E+01
-4.0000E+00  -1.6000E+01
-2.0000E+00  -4.0000E+00
 0.0000E-01   0.0000E-01
 2.0000E+00   4.0000E+00
 4.0000E+00   1.6000E+01
 6.0000E+00   3.6000E+01
 8.0000E+00   6.4000E+01
 1.0000E+01   1.0000E+02
```

MODEL I/P FROM LOG2:SPRNG.MOD COMPLETED

MODEL I/P FROM TTY: Status of model input channel reverts to Teletype

MODEL COMPLETE

*

Example 2

***LOG6:SPCON←I**

MODEL I/P FROM LOG6:SPCON.MOD

***CON**

CONFIGURATIONS :

NON-LINEAR SPRING WITH DASHPOT

```
 BLK  TYPE  B1  B2  B3
   4    /   17   6      ;Y DBLE DOT
   6    K
   9    I   48          ;    Y
  10    F    9
  17    W   48  10
  48    I    4          ;  Y DOT
```

MODEL I/P FROM LOG6:SPCON.MOD COMPLETED

MODEL I/P FROM TTY:

***LOG6:SPDAT←I**

MODEL I/P FROM LOG6:SPDAT.MOD

***PAR**

```
PARAMETERS :

  BLK        P1            P2            P3
    6  -5.0000E+00
    9  -1.0000E+01
   17   2.0000E+00   1.0000E+00

MODEL I/P FROM  LOG6:SPDAT.MOD COMPLETED

MODEL I/P FROM   TTY:

*LOG6:SPFUN<-I

MODEL I/P FROM  LOG6:SPFUN.MOD

*FUN

FUNCTIONS :

BLK NO.  10

COORD PAIRS :

-1.0000E+01  -1.0000E+02
-8.0000E+00  -6.4000E+01
-6.0000E+00  -3.6000E+01
-4.0000E+00  -1.6000E+01
-2.0000E+00  -4.0000E+00
 0.0000E-01   0.0000E-01
 2.0000E+00   4.0000E+00
 4.0000E+00   1.6000E+01
 6.0000E+00   3.6000E+01
 8.0000E+00   6.4000E+01
 1.0000E+01   1.0000E+02

MODEL I/P FROM  LOG6:SPFUN.MOD COMPLETED

MODEL I/P FROM   TTY:


MODEL COMPLETE

*
```

5.1.3 Block Output

While running the modelling program, the user may wish to make a number of separate alterations either to the specification statements or control parameters, and to re-execute the updated model each time. To preserve the block output for each run, the block output channel may be altered as in the following example:

```
*LOG1:OUT1<-O

BLOCK O/P   TO  LOG1:OUT1 .DAT

*
```

5.2 File Safety

5.2.1 Output Files

If the user attempts to overwrite a model or block output file that is already in existence on the disk, then the following message is typed on the Teletype by the program:

BLOCK/MODEL† *O/P TO LOG*n:"filename"."extension"

WILL OVERWRITE "filename"."extension" *ON DSK*
OK :

If the file is to be overwritten or written onto a device other than the disk, having previously assigned the device to the appropriate logical unit, the user types **Y** and the program will continue. Otherwise, the user types any other character, including a 'carriage-return'. The message

THEN CHANGE FILENAME

is then typed by the program and control is returned to the command mode. The status of the model or block output channel should then be altered by using the appropriate 'channel' command (see Section 5.1) before continuing. Block output and model output files are opened automatically following the 'GOE' and 'STORE' commands respectively.

Following a system failure during Batch processing, it may be possible that output files opened during the 'failed run' would need to be overwritten when the run is repeated, thus requiring interaction by the operator. To avoid this, when running the modelling program non-interactively (see Section 5.4), an output file will be overwritten automatically; in place of the above messages, the following message is typed by the program:

HAVE OVERWRITTEN FILE "filename"."extension" *ON DSK*

5.2.2 Input File

Immediately following a channel command that changes the status of the model input channel, the modelling program searches for the appropriate filename on the disk. If it cannot be found, the following message is typed by the program:

*MODEL I/P FROM LOG*n:"filename".*MOD*

"filename".*MOD NOT ON DSK*

IS IT ELSEWHERE :

If the file is to be read from a device other than the disk (see Section 5.1), the user types **Y** and the program will continue. Corresponding to the procedure with output files, by typing any other character, the message

THEN CHANGE FILENAME

is typed by the program and control returns to the command mode so that the input file status may be altered before continuing.

5.3 Modifying the Model

The specification statements stored in a file may be altered or added to using a text editing program such as 'TECO' or 'EDIT' before running the modelling program. If it is desired

---

† Whichever is appropriate.

to retain the comments following the dollar symbol or to retain an unsorted order of statements, it is necessary to make alterations in this way. Alternatively, specification statements may be altered or added to by repeating the appropriate command (i.e. 'TITLE', 'CON', 'PAR', or 'FUN') at any stage after the model has been set up; the modifications would usually be read in from the Teletype, but they may be read in from a file that is being used as the model input channel (see Sections 5.1 and 5.1.2). Alterations may also be made to any of the control parameters by repeating the appropriate command (i.e. 'INT' or 'OUT'), or to the status of the input and output channels. Following these alterations, the model is re-executed using the 'GOE' command.

5.4 Non-Interactive Running and Batch Processing

To run the modelling program in a non-interactive way, all the Teletype input that is typed by the user when running interactively must be stored in a file named BOMMP.IN. The modelling program first searches for a file with this name on the disk; if it is found, then the usual Teletype input is read from this file and most messages usually typed by the program are suppressed. This facility has been introduced primarily for Batch processing, but may also be used during On-Line processing to avoid tedious re-typing of the Teletype input by the user when it is desired to run a model that, except for minor modifications, has been run previously. When using this facility during On-Line processing, the messages typed by the program are shown in the following example:

```
RU BOMMP

NON-LINEAR SPRING WITH DASHPOT

** RUNNING **

RUN CPU TIME :  1.21 SEC.


END OF EXECUTION
CPU TIME: 2.62  ELAPSED TIME: 3.10
EXIT
```

The most convenient way to create a file containing the Teletype responses is to perform a 'dummy' run using the modelling program interactively with all quantities set to their desired value except for the upper time limit of integration, which is set much smaller than desired so that a minimum of CPU time is used. The Teletype responses typed by the user are stored by the program in 'LOG5:BOMMP.OUT'. The file BOMMP.OUT may then be renamed BOMMP.IN and the upper time limit of integration changed to the desired value using a text editing program. This method of creating the file provides the user with a check on the file format so that he can be sure the modelling program will run properly when used non-interactively.

To obtain the advantages of both non-interactive and interactive running during On-Line processing, the 'MANUAL' command has been introduced, which returns the program to interactive running and to the command mode. This enables alterations to be made interactively to a model that has been read in non-interactively and allows repeated model executions following further model alterations. Even if no alterations are envisaged, replacement of both the 'GOE' and 'EXIT' commands by a single 'MANUAL' command is advised and will require only the 'GOE' and 'EXIT' commands to be typed interactively by the user. If the 'MANUAL' command is typed while running interactively, e.g. when creating a file to be renamed BOMMP.IN by performing a dummy run (see above), control is returned to the command mode (i.e. in effect, nothing happens except that the command is written into the file BOMMP.OUT; see Section 5.1). On returning the program to interactive running by using the

'MANUAL' command, further Teletype input responses typed by the user are stored in the file FROMAN.OUT (on LOG5), and not in BOMMP.OUT.

5.5 Miscellaneous Commands

5.5.1 RUN

When setting up a model, the 'RUN' command may be used to initiate the command string 'TITLE;CON;PAR;FUN;INT;OUT;GOE'. If the Teletype is not used as the model input channel, the 'TITLE' command is excluded because the title is included with the configuration statements when previously stored using the 'STORE' command. Where part of the string has already been initiated by individual commands, the remaining ones are initiated. If required, the 'STORE' command may then be used following the execution. Having set up a model and executed it once, the 'RUN' command is equivalent to the 'GOE' command and could be used following alterations to the model by other commands.

If a file is used as the model input channel, the configuration (including the title), parameter, and function statements are all typed by the program in response to the 'RUN' command. The Teletype output of these statements may be terminated by typing "Control O". If all the model specification statements are not stored in the file used as the model input channel, a message indicating that a particular type of specification statement cannot be found is typed by the program and control is returned to the command mode. The model input channel should then be altered the appropriate number of times so that the remaining specification statements may be read in on repeating the 'RUN' command.

5.5.2 LOOK

Following the execution of a model, the output value of any block may be examined by using the 'LOOK' command. For the non-linear spring problem, an example of its use is as follows, control being returned to the command mode on typing a blank, or zero, block number:

***GOE**

*** RUNNING ***

RUN CPU TIME : 1.23 SEC.

***LOO**

BLK = **4**
O/P = *−1.0181E−01*

BLK = **9**
O/P = *1.6200E−01*

BLK = **10**
O/P = *3.2399E−01*

BLK = **17**
O/P = *5.0907E−01*

BLK = **48**
O/P = *9.2538E−02*

BLK = Blank block number returns control to command mode

*

5.5.3 DEBUG

For each block, the 'DEBUG' command provides the block output value X, the values X1, X2, and X3, the integer parameter M, and the parameter values P1, P2, and P3 (for notation, see Section 3.1). The above values are printed for each half time value during execution of the sort list. In response to prompts on the Teletype, the user specifies (*a*) the time interval during which the debugging output is required and (*b*) whether the output is required directly on the Teletype. If the response to (*b*) is anything other than Y, then the file named DEBUG.OUT is opened on LOG12 if it has not already been opened by a previous 'DEBUG' command. As with the input and output channels, the file DEBUG.OUT may be written onto a specific device other than the disk by assigning the appropriate device to LOG12 (see Section 5.1).

After the model has been executed, debugging for further runs is cancelled. The 'DEBUG' command must therefore be repeated prior to any further runs if debugging output is again required. To obtain debugging output at one time value only, the lower and upper limits should be made the same. For the non-linear spring problem, the use of the 'DEBUG' command is illustrated by the following example:

***DEB**

TIME LIMITS: LOWER, UPPER = **0.2, 0.3**

IS O/P TO TTY REQRD : **Y**

***GOE**

*** RUNNING ***

(See Appendix E for Teletype output)

RUN CPU TIME : 2.24 SEC.

*

5.5.4 PROGRAM

The 'PROGRAM' command enables another program (e.g. TRANS) to be run without specifically terminating the execution of BOMMP. Once execution of the new program is completed, control is returned to the monitor, not BOMMP. Following the 'PROGRAM' command, the message

PROGRAM NAME =

is typed on the Teletype by the program. The user responds by typing the name (maximum of five characters) of another program. Following a 'carriage-return', the new program is executed.

5.5.5 RETAIN

For each block with type I, TI, U, and Z (i.e. those requiring specified initial conditions), the 'RETAIN' command resets the initial condition (i.e. the first parameter) to the last calculated output value. When used following execution of the model, the 'RETAIN' command enables execution to be continued at a later stage during either the same program run or a subsequent one. In the latter case, the 'STORE' command would have to be used to retain the modified initial conditions in a file for future use.

6. LOOP BREAKING BLOCKS

When solving differential or difference equations, it is necessary to break the sort loop that occurs. Blocks whose output values are known at the beginning of each half time step enable this to be achieved. For differential equations, integrator and first order lag blocks are

used, while for difference equations, unit delay blocks are used. Only brief descriptions of these blocks are given in Table 1; further details are given below.

The vacuous (V) block is used to break the sort loop that occurs when solving an implicit equation. However, this block can only be used in conjunction with the wye (Y) block; its use is therefore described as a special facility in Section 8.1.

6.1 Unit Delay

Unit delay (U) blocks are entered in order of ascending block number at the beginning of the sort list following the constant (K) blocks and are the first to be executed at each step (see Section 3.2). This is because the input signal to a U block is calculated at the previous half time step (see definition in Table 1) so that it is not necessary for the input B1 to have already been sorted. Hence, provided the input of the U block is not itself another U block, the output value X of the U block is defined simply as the output value X1 of the input block B1. The procedure is more complex when the input to a U block is itself a U block; this occurs when the output value of a block at the previous complete time step is required. The validity of the above definition of the function performed by a U block then depends on the order of entry of the U blocks in the sort list. To avoid this difficulty, the U blocks are therefore executed in the following two stages, with the first stage being completed for all the U blocks before commencing the second stage immediately after:

(*a*) *Stage 1*
When $t = t_0$, X = P1
When $t > t_0$,
X = X1 if B1 is not a U block
X = P2 if B1 is a U block
(Both X1 and P2 are last calculated at previous half time step)

(*b*) *Stage 2* (for all values of t)
No further calculation if B1 is not a U block
P2 = X1 if B1 is a U block (P2 is then used in Stage 1 at next half time step)

Consider a function y, which is calculated in a user-defined block, U3 say, and assume a unit time interval (or step size) and that the current step value is k (i.e. output of the U3 block is $y(k)$). The following configuration statements show how y may be represented at each of the previous two steps:

3	U	5	$y(k-2)$
5	U	6	$y(k-\frac{3}{2})$
6	U	4	$y(k-1)$
4	U	2	$y(k-\frac{1}{2})$
2	U3	9	$y(k)$

6.2 Integrator

Consider the first order differential equation

$$dy/dt = f[t, y]$$

Then, for the second order Runge Kutta (i.e. Modified Euler) method used in the integrator (I) block, the dependent variable y, which is the output value of the I block, is defined as

$$y(t+h/2) = y(t) + (h/2) f[t, y(t)]$$

$$y(t+h) = y(t) + h f[t+h/2, y(t+h/2)]$$

where h is the time step. It can be seen that at each half time step, y is defined in terms of quantities calculated previously. It is therefore evaluated at the beginning of each half time step before execution of the sort list (see Section 3.3). Like most other blocks though, an I block is entered into the sort list only when its inputs have already been sorted. On executing an I block in the

sort list, the function f is calculated so that y may be derived at the beginning of the next half time step as described above.

The first order lag (T1) block, described in Section 6.3, uses the same integration procedure as the I block. These blocks together are numbered in the order in which they appear in the sort list; this number is placed in the integer parameter M for each of these blocks, thus providing an order for the evaluation of y prior to entry to the sort list at each half time step.

The facility to reset y to its initial condition or hold it at its current value has been included in the I block by using the values X2 and X3 as switches[7] (see Table 1). For the reset operation, the value of y calculated prior to execution of the sort list is overwritten with the reset value. It may therefore be important to ensure that the integration block will be sorted before being used as an input to another block (see Section 3.2). For the hold operation, the value of dy/dt is set equal to zero so that the current value of y is unaltered.

6.3 First Order Lag

The output of the first order lag (T1) block is the solution of the differential equation

$$y + \mathrm{P2}\, dy/dt = \mathrm{X1} + \mathrm{X2} + \mathrm{X3}$$

for an initial value of y equal to P1. Except for the hold and reset capability, which the T1 block does not have, T1 blocks are treated in exactly the same way as I blocks; they are sorted and executed in the same way and use the same method of numerical integration. The use of the associated integer parameter M is described in Section 6.2.

7. USER-DEFINED BLOCKS

Complex algebraic expressions, which may include special functions such as trigonometric or exponential functions, can only be practicably handled in a 'block oriented' language by use of user-defined blocks. In CSMP-10(ARL), these blocks have therefore been introduced and are identified in the configuration statements by a block type of the form Um, where m is an integer not exceeding fifteen; this number may be increased by simple coding alterations to the modelling program. Each block may have up to three parameters and three sorted block inputs (as declared in the configuration statement), but may have any number of unsorted block inputs. When reference is made to a block of type Um, the quantities required to be calculated must be defined in a FORTRAN subroutine named USERm. However, each of these subroutines may refer to other FORTRAN subroutines or functions. All 'user-defined' subroutines and 'other' associated subroutines and functions are compiled and loaded into core with the compiled modelling program (see Section 4.1.1).

As well as the output value X of the user-defined block (referred to as the 'principal' output), 'additional' outputs may be specified within the subroutines by use of 'user output' blocks. These are identified in the configuration statement by a block type 'UO' and an input B1 that is the associated user-defined block number. They are not entered in the sort list, but are listed following their associated user-defined block when storing the model (see Section 3.2).

Because use may be made in these subroutines of variables stored by the modelling program in labelled COMMON storage areas, a description of these variables in given in Appendix F.

For an understanding of the rest of this section, the reader should refer to the subroutine example given in Appendix G. A model incorporating this subroutine is described at the end of this section.

The same user-defined block type may be used in more than one configuration statement. Within user-defined subroutines therefore, it is advisable to avoid referring to specific block numbers for both the principal and additional outputs. This may be achieved in the following manner. When reference is made to a user-defined subroutine, the block number B of the appropriate user-defined block is stored in 'L', which is one of the subroutine arguments (see Appendix G). Outputs of each block used in the model are stored in the array 'C' so that the principal output should be set in the array element C(L). Additional elements should then be set in consecutive elements C(L + 1), C(L + 2), . . . C(L + NUO), where NUO is the number of

these additional outputs. An examples of a set of appropriate configuration statements (for NUO = 3) is as follows:

```
12  U2   8  5     Principal output of subroutine USER2
13  UO  12        First additional output of subroutine USER2
14  UO  12        Second additional output
15  UO  12        Third additional output
```

User-defined subroutines enable the user to overwrite the output X, inputs B1, B2, B3, and parameters P1, P2, P3 of any block. As an important safeguard in using UO blocks, it is therefore important to test for compatibility between the user-defined subroutine and appropriate UO configuration statement so that the output of another block is not inadvertently overwritten.

For UO block number N, the tests are:

(1) the block is defined as a UO block, i.e. MTRX(5*N − 4) = 3 (the type number of a UO block, see Table 1); and
(2) B1 is the associated user-defined subroutine block number, i.e. MTRX(5*N−3) = L (i.e. block number B).

These tests need only be performed at the initial time step, i.e. when TEST(5) = 1 (see Appendix F), and are performed in subroutine UOTEST (see Section 9.1), which is part of the basic modelling program. For NUO consecutive UO blocks, the appropriate statements are given in the example of Appendix G; for a non-consecutive UO block number N, the test statement is

```
IF(TEST(5).EQ.1)CALL UOTEST(N,L,MTRX)
```

The first four FORTRAN statements in Appendix G would generally be included in all user-defined subroutines, with the appropriate value for m in the subroutine name 'USERm'. For convenience, B1, B2, B3, X1, X2, X3, P1, P2, and P3 are obtained in Appendix G from the appropriate arrays for the particular user-defined block.

Though only three inputs and three parameters are allowed in each user-defined block, for all other blocks, the output X, inputs B1, B2, B3, and parameters P1, P2, P3 may be obtained from the arrays 'C', 'MTRX', and 'PAR' (see Appendix G for definition of arrays). However, a knowledge of the sorting process (see Section 3.2) is necessary to determine whether the block output values are at the present or previous half time step. For each UO block, the parameters P1, P2, P3 and the inputs B2 and B3 may be used, but the latter inputs are not required to be sorted, since UO blocks are not entered in the sort list.

When large user-defined subroutines are envisaged, it is advisable to construct a small user-defined subroutine that refers to other normal FORTRAN subroutines and functions. Any block number changes may then be confined to the small user-defined subroutine, thus avoiding time consuming recompilation of large subroutines.

The user-defined subroutine in Appendix G enables the divider, constant, and weighted summer blocks used in the model of the non-linear spring problem to be replaced by a user-defined block of type U1 (see Figure 3). In addition, a user output block provides an additional output whose value is the same as the replaced weighted summer block. It is not suggested that the above replacement is necessarily desirable; it is made solely for the purpose of presenting a simple example of the use of user-defined subroutines. Given that the model for the non-linear spring problem has been set up and that the integration and output control parameters have been specified (i.e. the model is at the completion of Stage 6 in Section 4.1), the appropriate modifications are given below:

```
*CON

CONFIGURATIONS :

BLK, TYPE, B1, B2, B3
```

```
4, U1, 48, 10; Y DBLE DOT
PREV BLK   4 DELETED
5, UO, 4            $ B*(Y DOT)+F(Y)
6
PREV BLK   6 DELETED
17
PREV BLK  17 DELETED

*PAR

PARAMETERS :

BLK, P1, P2, P3
4, 5, 2

*FUN

FUNCTIONS :

BLK NO. =

MODEL COMPLETE

*
```

At this stage, the model may be run using the 'GOE' command; the results obtained will be identical to those obtained previously.

8. SPECIAL FACILITIES

The two special facilities described here are the solution of an implicit equation and the representation of discrete systems such as difference equations. The two blocks, wye (Y) and vacuous (V), required for the solution of an implicit equation can only be used together for that purpose, whereas the three blocks, unit delay (U), zero order hold (Z), and time pulse generator (T), required in discrete system simulation may be used separately in other applications (see Table 1 for their definitions).

8.1 Solution of an Implicit Equation

An implicit equation (e.g. $y = f(y)$, where f is a function of y and possibly other variables, e.g. time, that are independent of y) cannot be solved directly in CSMP-10(ARL) since the output value of the block that calculates f is required to be its own input signal; the resulting loop would thus lead to a failure of the sort algorithm. However, by using Y and V blocks (see Table 1), an implicit equation may be solved. The V block 'breaks' the sort loop, and the Y block performs the numerical iteration process for calculating y; the final value for y is stored as the output value of the Y block.

Assuming that the function f is calculated in a user-defined block, U6 say, Figure 4 shows the block structure necessary to solve the implicit equation $y = f(y)$. The appropriate block numbers shown correspond to the following set of configuration statements:

```
3   Y    2   7        y
2   U6   7            f(y)
7   V
```

Any other blocks that are required to calculate f must also have the output value of the V block as an input signal. The iterative method[10] used to solve the implicit equation $y = f(y)$ is described in Appendix H. Included in Appendix H is a table of all the quantities printed for each iterative cycle when using the debugging facility (see Section 5.5.3).

It should be realised that the numerical method used to solve an implicit equation will not achieve convergence for all implicit equations. Very few numerical techniques are capable of solving all problems of a particular type. The method used in CSMP-10(ARL), which is a significant improvement on the somewhat crude method used in CSMP-10, should be viewed as a useful technique that will in most cases allow the user to solve an implicit equation without having to program his own numerical method in a user-defined subroutine.

8.2 Discrete System Simulation

Discrete systems, such as difference equations, may be modelled in CSMP-10(ARL) by using T, Z, and U blocks (see Table 1). Consider the second order difference equation

$$y(m) = f[y(m-1), y(m-2)] \quad \text{for } m = 2, 3, \ldots$$

where the values $y(0)$ and $y(1)$ are known, and the function f is specified in a user-defined block, U3 say. CSMP-10(ARL) is written primarily for continuous systems with time, t, as the independent variable. When dealing with purely discrete systems, the time step may be regarded as a cycle step; this is achieved by setting the pulse train period (in the T block) equal to the time step. For simplicity in the following description, these quantities will be assumed to be one second, which corresponds to a unit cycle step.

The combination of a Z, U, and Z block in series is such that if the output value of the first Z block is $y(m)$, then the output value of the second Z block is $y(m-1)$. Similarly, a Z, U, Z, U, and Z block series combination may represent $y(m)$, $y(m-1)$, and $y(m-2)$ as the output value of the first, second, and third Z blocks respectively. Figure 5 shows the block structure necessary to represent the above difference equation. The appropriate block numbers shown correspond to the following set of configuration statements:

```
4    T    1          $ PULSE TRAIN      Block 1 = t
6    Z    10   4                         y(m)
7    Z    12   4                         y(m-1)
8    Z    13   4                         y(m-2)
12   U    6
13   U    7
10   U3   8    7                         f[y(m-1), y(m-2)]
```

The output value of each block used, in execution order, is described in Table 3 for each half cycle step up to the second complete step. Following the initial step (i.e. $k = 0$), the output value of each U block remains the same as the Z block that follows in the series combination of

TABLE 3

BLOCK OUTPUT VALUES FOR A SECOND ORDER DIFFERENCE EQUATION EXAMPLE

Block number	Type	Step (k)				
		0	$\frac{1}{2}$	1	$\frac{3}{2}$	2
12	U	$y(1)$	$y(2)$	$y(2)$	$y(3)$	$y(3)$
13	U	$y(0)$	$y(1)$	$y(1)$	$y(2)$	$y(2)$
4	T	1	0	1	0	1
7	Z	$y(1)$	$y(1)$	$y(2)$	$y(2)$	$y(3)$
8	Z	$y(0)$	$y(0)$	$y(1)$	$y(1)$	$y(2)$
10	U3	$y(2)$	$y(2)$	$y(3)$	$y(3)$	$y(4)$
6	Z	$y(2)$	$y(2)$	$y(3)$	$y(3)$	$y(4)$

Z and U blocks (see Table 3). By setting the U block parameter values in the way shown in Figure 5 this property also applies at the initial step. In order to obtain the correct output value of the first Z block (i.e. block number 6) at $k = 0$, $y(2)$ should be calculated using the difference equation and used as the parameter P1 for the Z block.

9. DESCRIPTION OF MODELLING PROGRAM

The computer program structure and storage requirement are briefly described below. Further information may be obtained by referring to the FORTRAN listing.

9.1 Program Structure

The computer program consists of a small main program (MAIN.) used to open the arrays that are later expanded, a number of major subroutines, various service subroutines and functions, and a BLOCK DATA subprogram which specifies the list of block type names (see Appendix I).

9.2 Storage Requirement

Because arrays are expanded to the required size by the modelling program (see Section 4.1.2), it may be necessary to know the storage dependence on the number of various blocks specified. When loaded, the basic modelling program (see Section 4.1.2), without any additional subroutines specified by the user, requires 12K words of core store (1K = 1024). On expanding the arrays, the number of additional words required is given approximately by the formula

$$13\ \text{NBLK} + 4\ \text{NINT} + \text{NDELAY} + 31\ \text{NFUN}$$

where the FORTRAN integer quantities NBLK, NINT, NDELAY, and NFUN are defined in the COMMON statement labelled 'SIZE' in Appendix F and are the quantities typed (in the above order) by the user in Section 4.1.2.

10. CONCLUDING REMARKS

A description of the 'block oriented' simulation language CSMP-10(ARL) has been given. The language, which has been developed from CSMP-10, is written mainly in FORTRAN IV for a PDP-10 computer. Three major improvements have been made. Firstly, 'user-defined' blocks written as FORTRAN subroutines have been included. These blocks may have only three sorted inputs, but may have any number of unsorted inputs. 'Dummy' or 'user-output' blocks have been created, whose output value is defined in the appropriate user-defined subroutine, thus allowing any number of outputs for each of these blocks. Secondly, arrays necessary to store information on each block used are automatically expanded to the size determined by the user. Thirdly, the language has been divided into a modelling program, which is used to perform the model simulation and store the output in a binary file on a specified storage device, and an output program, which is used to print and plot the character conversion of the binary file. Although the modelling program is designed to be run interactively from a Teletype, it may also be run non-interactively during Batch or On-Line processing. A large number of minor improvements have been made, many of them being corrections to errors in the available version of CSMP-10. The Teletype messages and responses have been completely revised, and much greater detail has been provided here than in descriptions of previous versions of the langugae

ACKNOWLEDGEMENT

The authors are indebted to the assistance provided by T. J. Packer of the Weapons Research Establishment, Salisbury, South Australia and to Dr. L. H. Mitchell of the Computer Centre at ARL. Mr. Packer suggested the development of CSMP-10 to enable the simulation of helicopter flight manoeuvres and provided the subroutines used in the improved integration facilities. Dr. Mitchell provided much valuable advice in the early stages, particularly in implementing the major improvements.

REFERENCES

1. Carnegie Mellon University — 'CSMP-10'. DECUS Program No. 10-122 (1971).

2. Brennan, R. D. and Sano, H. — 'PACTOLUS—A Digital Analog Simulator Program for the IBM 1620'. Proceedings, 1964 Fall Joint Computer Conference, AFIPS vol. 26, pp. 299–312.

3. IBM Application Program — '1130 Continuous System Modelling Program (1130-CX-13X)" Application Description, H20-0209-1.

4. IBM Application Program — '1130 Continuous System Modelling Program (1130-CX-13X)'. Program Description and Operations Manual, H20-0282-1.

5. Carnegie Mellon University — 'Block CSMP-9, A Block Oriented Continuous System Modelling Program for the PDP-9'. User Manual (1969).

6. Packer, T. J. — '1130 CSMP—Extended for 16K Configuration'. Report, Faculty of Engineering, University of Singapore, 1967.

7. Packer, T. J. — 'An extended Version of the 1130 CSMP'. SIMULATION, vol. 9, no. 6, December 1967.

8. Computer Sciences Corporation — '***CSMP Continuous System Modelling Program'. CSCX Basic Library, Program No. 01-3199, July 1971.

9. Nankivell, P. G. and Gilbert, N. E. — 'A General Purpose Output Program for Use in Simulation'. ARL Note 367, December 1976.

10. Wegstein, J. H. — 'Accelerating Convergence of Iterative Processes'. Comm. ACM, June 1958.

Postal Address: Chief Superintendent, Aeronautical Research Laboratories, P.O. Box 4331, Melbourne, Victoria 3001

APPENDIX A

Time Pulse Generator Algorithm

The first pulse occurs when X1 $\geqslant 0$, and a pulse train of period P1 continues until X1 < 0 (if at all), but is restarted if subsequently X1 $\geqslant 0$. Consider the case when X1 $\geqslant 0$ at $t = t_1$; then

$$\begin{aligned} X &= 0 \text{ if } t < t_1 \\ &= 1 \text{ if } t_1 + n\text{P1} \leqslant t < t_1 + (n+0{\cdot}5)\text{P1} \\ &= 0 \text{ if } t_1 + (n+0{\cdot}5)\text{P1} \leqslant t < t_1 + (n+1)\text{P1} \end{aligned}$$

where $n = 0, 1, 2, \ldots$, and provided X1 does not become negative. The parameter P2 and integer parameter M are used in the time pulse generator algorithm, which is described as follows (h is the time interval for integration—defined in Section 3.1):

(a) Initial entry

M = 0; go to (1)

(b) Subsequent entries

(1) If X1 $\geqslant 0$, go to (2)
X = 0; M = 0; exit

(2) If M = 1, go to (3)
X = 1; M = 1; if P1 $\leqslant 0$, print appropriate diagnostic message (see Appendix B) and exit; P2 = (h−P1)/2+0·0001*min(h, P1); exit

(3) If P2 < 0, go to (4)
Change X (= 0 or 1); P2 = P2−0·5*P1; go to (3)

(4) P2 = P2+0·5*h; exit

The quantity min(h, P1) is equal to either h or P1, whichever is the smallest; the quantity 0·0001*min (h, P1) is therefore an error term that ensures, when using floating point arithmetic in a computer, that the equality for t occurs at the lower limit (see definition of X above).

APPENDIX B

Modelling Program Diagnostic Error Messages

To distinguish the following error messages from the FORTRAN compiler diagnostics, all the messages below are preceded by an exclamation mark (!). Integer constants are represented by n and 's.e.' is used to abbreviate 'self-explanatory'.

! ATTEMPT TO DIVIDE BY ZERO IN BLK n —when using divider block

! ATTEMPT TO TAKE SQRT OF A –VE NO. IN BLK n —when using half power block

! BLK O/P CANNOT GO TO TTY —since in binary form

! BLK 1 RESERVED FOR TIME —s.e.

! BLK n *HAS AN ILLEGAL BLK TYPE* ("type") —see Table 1 for legal block types

! BLK n *IS NOT A UO BLK* —when attempting to define an additional output in a user-defined subroutine

! BLK n *IS NOT AN F BLK* —s.e.

! BLK n *NOT USED* —when specifying output blocks

! CODE PRIOR TO CALL TO EXPAND UNRECOGNISED —re-load the program; if error persists, keep data files and consult originator

! COMMAND ERROR —invalid program command (see Table 2*a*)

! COMMAND NO LONGER VALID —for command used in earlier version

! CONFIGS ALREADY READ FROM MODEL I/P FILE —s.e.

! CONFIGS NOT COMPLETE —s.e.

! ERROR DURING MODEL EXECUTION —s.e.

! ERROR IN BLK n —invalid configuration statement (see Table 1)

! EXTENSIONS ALREADY SET —extensions, including period, should not be included in 'channel' command

! FATAL ATTEMPT TO READ NON-EXISTENT DSK FILE "filename"."extension" —causes program to crash; may occur only when running non-interactively

! FINAL TIME MUST NOT BE LESS THAN INITIAL TIME —s.e.

! FUNCTS ALREADY READ FROM MODEL I/P FILE —s.e.

! FUNCTS NOT COMPLETE —s.e.

! ILLEGAL DEVICE NAME —must be LOGn, where n = 1 to 12 (excluding 5), or TTY

! *IMPROPER PARAM SPECIFICATION* —s.e.

! *INCORRECT RECORD . . . . .* "record" **** —try again

! *INTEGN CONTROL PARAMS NEEDED* —s.e.

! *INTEGN INTERVAL MUST NOT BE GREATER THAN TIME RANGE* —s.e.

! *INVALID BLK NO.* —s.e.

! *ITERATION LIMIT OF* n *EXCEEDED FOR V & Y BLKS* n *&* n *AT TIME t* — see Appendix H

! *I/P (BLK* n) *TO BLK* n *IS NOT DEFINED* —s.e.

! *I/P BI FOR U0 BLK* n *IS NOT THE ASSOCIATED USER-DEFINED BLK* n —when the user output configuration statement is imcompatible with the corresponding user-defined statement

! *I/P B2 FOR Y BLK* n *IS NOT A V BLK* —see Figure 4

! *LOGICAL UNIT* n *ALREADY IN USE* —by another channel

! *MAX OF* n *FUNCTS EXCEEDED* —when specifying function coordinates

! *MAX OF* n *F BLKS EXCEEDED* —when specifying configuration statements

MAX OF n *I & TI BLKS EXCEEDED* —s.e.

! *MAX OF* n *U BLKS EXCEEDED* —s.e.

! *MAX OF* n**15 COORD PAIRS EXCEEDED* —see Section 3.1.3

! *MODEL NOT COMPLETE* —all configuration, parameter, and function statements must first be set

! *MORE WORK TO DO* —attempt to execute model before setting integration or output control parameters

! *NO CORRESPONDING CONFIG STATEMENT* —when specifying parameters

! *NO CONFIGS IN MODEL I/P FILE* —s.e.

! *NO FUNCTS IN MODEL I/P FILE* —s.e.

! *NO PARAMS IN MODEL I/P FILE* —s.e.

! *NOT ENOUGH CORE TO SET UP ARRAYS* —when expanding arrays in **'EXPAND'**

! *O/P CONTROL PARAMS NEEDED* —s.e.

! *PARAMS ALREADY READ FROM MODEL I/P FILE* —s.e.

! *PERIOD (PI) FOR T BLK* n *MUST BE GREATER THAN ZERO* —see Appendix A

! *PNT WITH ABSCISSA* 'real constant' *DOES NOT EXIST* —when attempting to delete a coordinate pair

! *SORT FAILURE AT BLK* n —see Section 3.2

! *SYNTAX ERROR* —typing error or inappropriate response to Teletype message typed by the program

! *USER-DEFINED SUBROUTINE NOT LOADED FOR BLK* n —when the user has not loaded the appropriate user-defined subroutine or subroutine used for user-defined interpolation in the function block

APPENDIX C

User-Defined Interpolation for Function Block

The modelling program allows the user to specify up to three separate interpolation or curve-fitting (e.g. using least squares technique) formulae for use in the function (F) block. The appropriate formula to be used for each F block is determined by the values set for P1. Corresponding to P1 = −1, −2, and −3, the formula used must be programmed in the FORTRAN subroutines INTRP1, INTRP2, and INTRP3 respectively. The first two statements of these subroutines (e.g. for INTRP2) take the form

```
SUBROUTINE INTRP2 (L, C, MTRX, PAR, YOUT, XIN, MFIRST, MLAST, F)
DIMENSION C(1), MTRX(1), PAR(1), F(1)
```

where L, C, MTRX, and PAR are defined in Appendix G,
YOUT is the function output value set by the user,
XIN is the independent variable (abscissa) for which YOUT is required, and
F is an array containing the coordinate pairs X(K), Y(K)
such that X(K) = F(2*K − 1) and Y(K) = F(2*K)
for K = MFIRST, MFIRST + 1, . . ., MLAST

For each F block, the parameter P3 is used to store MFIRST, and the integer parameter M is used to store the number of coordinate pairs, *i.e.* *MLAST–MFIRST + 1*

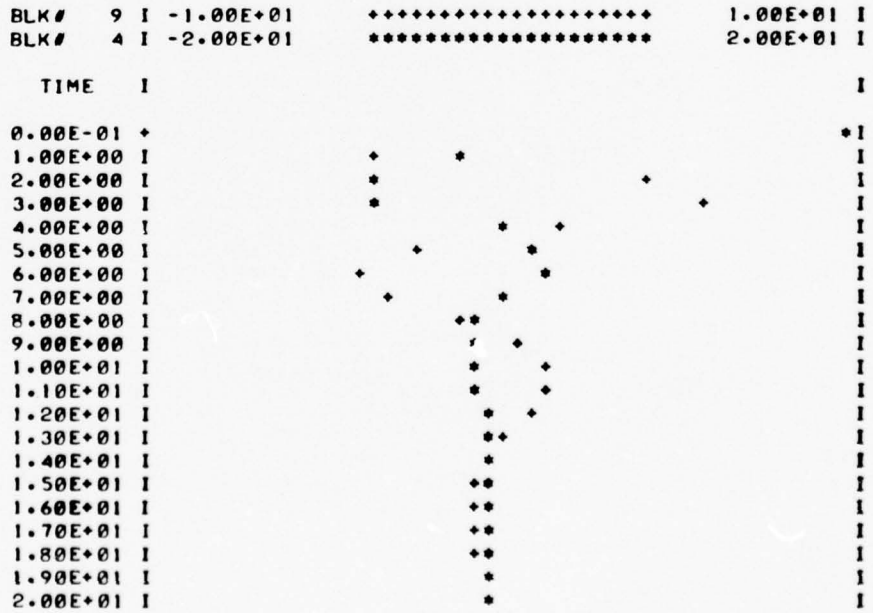
APPENDIX D

Block Output for Non-Linear Spring Problem

(1) Tabular Output

TIME	BLK# 9	BLK# 4	BLK# 48	BLK# 10
0.00E-01	-1.00E+01	2.00E+01	0.00E-01	-1.00E+02
1.00E+00	-3.27E+00	-1.42E+00	9.36E+00	-1.16E+01
2.00E+00	4.50E+00	-6.41E+00	5.52E+00	2.10E+01
3.00E+00	6.06E+00	-6.39E+00	-2.44E+00	3.68E+01
4.00E+00	1.86E+00	1.10E+00	-4.62E+00	3.71E+00
5.00E+00	-2.02E+00	2.01E+00	-2.98E+00	-4.10E+00
6.00E+00	-3.68E+00	2.90E+00	-2.03E-01	-1.41E+01
7.00E+00	-2.69E+00	8.95E-01	1.84E+00	-8.15E+00
8.00E+00	-7.47E-01	-4.38E-01	1.84E+00	-1.49E+00
9.00E+00	8.00E-01	-7.97E-01	1.19E+00	1.60E+00
1.00E+01	1.58E+00	-7.82E-01	3.74E-01	3.16E+00
1.10E+01	1.60E+00	-5.23E-01	-2.92E-01	3.20E+00
1.20E+01	1.10E+00	-1.83E-01	-6.46E-01	2.20E+00
1.30E+01	4.17E-01	1.04E-01	-6.76E-01	8.33E-01
1.40E+01	-1.75E-01	2.63E-01	-4.81E-01	-3.51E-01
1.50E+01	-5.16E-01	2.85E-01	-1.97E-01	-1.03E+00
1.60E+01	-5.80E-01	2.09E-01	5.66E-02	-1.16E+00
1.70E+01	-4.37E-01	9.16E-02	2.08E-01	-8.74E-01
1.80E+01	-2.02E-01	-1.64E-02	2.43E-01	-4.05E-01
1.90E+01	1.96E-02	-8.35E-02	1.89E-01	3.92E-02
2.00E+01	1.62E-01	-1.02E-01	9.25E-02	3.24E-01

(2) Graphical Output



APPENDIX E

Debugging Output for Non-Linear Spring Problem

TIME	B	T M	X	X1 P1	X2 P2	X3 P3
2.0000E-01	9	I	-9.6097E+00	3.7726E+00	0.0000E-01	0.0000E-01
		1		-1.0000E+01	0.0000E-01	0.0000E-01
2.0000E-01	10	F	-9.2975E+01	-9.6097E+00	0.0000E-01	0.0000E-01
		11		0.0000E-01	0.0000E-01	1.0000E+00
2.0000E-01	17	W	-8.5430E+01	3.7726E+00	-9.2975E+01	0.0000E-01
		0		2.0000E+00	1.0000E+00	0.0000E-01
2.0000E-01	4	/	1.7086E+01	-8.5430E+01	-5.0000E+00	0.0000E-01
		0		0.0000E-01	0.0000E-01	0.0000E-01
2.0000E-01	48	I	3.7726E+00	1.7086E+01	0.0000E-01	0.0000E-01
		2		0.0000E-01	0.0000E-01	0.0000E-01
2.5000E-01	9	I	-9.4211E+00	4.6269E+00	0.0000E-01	0.0000E-01
		1		-1.0000E+01	0.0000E-01	0.0000E-01
2.5000E-01	10	F	-8.9580E+01	-9.4211E+00	0.0000E-01	0.0000E-01
		11		0.0000E-01	0.0000E-01	1.0000E+00
2.5000E-01	17	W	-8.0326E+01	4.6269E+00	-8.9580E+01	0.0000E-01
		0		2.0000E+00	1.0000E+00	0.0000E-01
2.5000E-01	4	/	1.6065E+01	-8.0326E+01	-5.0000E+00	0.0000E-01
		0		0.0000E-01	0.0000E-01	0.0000E-01
2.5000E-01	48	I	4.6269E+00	1.6065E+01	0.0000E-01	0.0000E-01
		2		0.0000E-01	0.0000E-01	0.0000E-01
3.0000E-01	9	I	-9.1470E+00	5.3791E+00	0.0000E-01	0.0000E-01
		1		-1.0000E+01	0.0000E-01	0.0000E-01
3.0000E-01	10	F	-8.4647E+01	-9.1470E+00	0.0000E-01	0.0000E-01
		11		0.0000E-01	0.0000E-01	1.0000E+00
3.0000E-01	17	W	-7.3888E+01	5.3791E+00	-8.4647E+01	0.0000E-01
		0		2.0000E+00	1.0000E+00	0.0000E-01
3.0000E-01	4	/	1.4778E+01	-7.3888E+01	-5.0000E+00	0.0000E-01
		0		0.0000E-01	0.0000E-01	0.0000E-01
3.0000E-01	48	I	5.3791E+00	1.4778E+01	0.0000E-01	0.0000E-01
		2		0.0000E-01	0.0000E-01	0.0000E-01

APPENDIX F

Modelling Program Common Variables

For each labelled COMMON statement, the FORTRAN variables appear in the order in which they are described; e.g. for COMMON storage area labelled 'ODEVIM', the COMMON statement is COMMON/ODEVIM/ONUM, ODEV, IDEV.

COMMON label	FORTRAN identifier	Declared type	Description
BATCH	BATCH	logical	BATCH = .TRUE. if a file named 'BOMMP. IN' is found on the disk = .FALSE. otherwise
BUFF	NBUFF	integer	Last word number filled in array 'BUFFER'
	BUFFER(126)	real	Block output buffer which is written into a binary file only when NBUFF $\geq$ 127
	NFLAG	integer	NFLAG = 1 prior to output of first block value = 0 otherwise
	XPC	real	Percentage change required for output, divided by 100 (i.e. relative change)
DEBUG	DEBUG	logical	DEBUG = .TRUE. if debugging in operation = .FALSE. otherwise
	DMAX	real	Upper time limit for debugging
	DMIN	real	Lower time limit for debugging
	LOGIND	integer	LOGIND = 'Y' for debugging output directly on the teletype
EXTRA2	TY(48)	real	List of block type names (including U1 to U15)
FILE	PDEV	integer	Model output device name
	IFILNA(2)	integer	Model input filename
	OFILNA(2)	integer	Block output filename
	PFILNA(2)	integer	Model output filename
FIRST	FIRST	logical	FIRST = .TRUE. for block output at initial time = .FALSE. subsequently
LINESP	LINESP(1)	integer	LINESP(1) = '(1H)'; array containing format specification to correct line spacing following the change in the FORTRAN operating system from 'FORSE' to 'FOROTS'
NEWFIL	NEWFIL	integer	NEWFIL = 0 if block output file has not been written onto = 1 otherwise
NUMB	NOD	integer	Number of U blocks used
	NEQ	integer	Number of I and T1 blocks used
	NFG	integer	Number of F blocks used
	NCON	integer	Number of K blocks used
	NLIST	integer	Total number of blocks used

APPENDIX F (continued)

COMMON label	FORTRAN identifier	Declared type	Description
NUMBA	NUMBA	integer	Number of blocks to be output
ODEVIM	ONUM	integer	Block output logical unit number
	ODEV	integer	Block output device name, i.e. LOGn
	IDEV	integer	Model input device name
PDEVIM	PNUM	integer	Model output logical unit number
SIZE	NBLK	integer	Maximum block number
	NINT	integer	Maximum number of I and TI blocks
	NFUN	integer	Maximum number of F blocks
	NDELAY	integer	Maximum number of U blocks
	NPNTS	integer	Number of coordinate pairs used in F blocks
TEST	TEST1	integer	TEST1 = 1 if model input file not opened = 2 if opened, but not completely read = 0 if completely read
	TEST2	integer	Model input logical unit number, if equal to 5, Teletype used as model input channel
	TEST3	integer	TEST3 = 2 if functions have been read in = 1 otherwise
	TEST4	integer	TEST4 = 1 if no F blocks in the model = 2 if data for F blocks not completely read in = 3 if completely read in
	TEST5	integer	TEST5 = 1 for initial time step = 2 for each half time step = 3 for each complete time step = 4 for error during execution = 5 for run terminated by typing "↑" = 6 for run terminated by a Q block
	TEST6	integer	Number representing location of last character in input record read in; when TEST6 = 0 or TEST6 = 72, new input record is read in
	TEST7	integer	TEST7 = 1 if integration parameters not read in = 2 otherwise
	TEST8	integer	TEST8 = 1 if output control parameters not read in = 2 otherwise
	TEST9	integer	TEST9 = 1 if sort process not completed = 2 if sort process successfully completed
TIM	DT	real	Integration interval h
	DTS2	real	$h/2$
	TTOT	real	Final time
	TZERO	real	Initial time
TITLE	TITLE(12)	real	Title (up to 60 characters)
V	TSAMP	real	Output time interval
	T	real	Time value, t

APPENDIX G

Example of User-Defined Subroutine

```
      SUBROUTINE USER1 (L,C,MTRX,PAR)
      INTEGER TEST,B1,B2,B3
      COMMON/TEST/TEST(9)
      DIMENSION C(1),MTRX(1),PAR(1)
C
C ***************************************************************
C *
C *  L IS THE BLOCK NUMBER OF THE USER-DEFINED BLOCK; THE SAME
C *       SUBROUTINE COULD BE USED BY OTHER BLOCKS.
C *
C *  C(N) IS THE OUTPUT OF BLOCK N AT THE CURRENT TIME; C(L) MUST
C *       THEREFORE BE SET EQUAL TO THE PRINCIPAL OUTPUT OF THE
C *       USER-DEFINED SUBROUTINE.  SINCE TIME IS STORED IN
C *       BLOCK 1, C(1) IS THE TIME VALUE.
C *
C *  MTRX IS AN ARRAY SUCH THAT FOR BLOCK N
C *       MTRX(5*N-4) = TYPE NUMBER
C *       MTRX(5*N-3) = B1
C *       MTRX(5*N-2) = B2
C *       MTRX(5*N-1) = B3
C *         MTRX(5*N) = QUANTITY DEPENDENT ON BLOCK TYPE AND
C *                     INSERTED BY THE MODELLING PROGRAM
C *
C *  PAR IS AN ARRAY SUCH THAT FOR BLOCK N
C *      PAR(3*N-2) = P1
C *      PAR(3*N-1) = P2
C *        PAR(3*N) = P3
C *
C *  TEST(5) = 1 FOR INITIAL TIME STEP
C *          = 2 FOR EACH HALF TIME STEP
C *          = 3 FOR EACH COMPLETE TIME STEP
C *          = 4 FOR ERROR DURING EXECUTION
C *
C *  OTHER ELEMENTS IN ARRAY 'TEST' ARE NOT REQUIRED IN THIS
C *  EXAMPLE
C *
C ***************************************************************
C
C  NUO = NUMBER OF CONSECUTIVE UO BLOCKS
C
      DATA NUO/1/
C
C  TEST FOR COMPATIBILITY OF UO BLOCKS WITH USER-DEFINED BLOCK
C  TEST(5) IS SET EQUAL TO 4 IN SUBROUTINE UOTEST IF INCOMPATIBLE
C
      IF((TEST(5).GT.1).OR.(NUO.LT.1))GO TO 20
      DO 10 J=1,NUO
      N=L+J
10    CALL UOTEST (N,L,MTRX)
      IF(TEST(5).EQ.4)RETURN
```

```
C
C  INPUT VALUES AND PARAMETERS OF USER-DEFINED BLOCK
C
20      B1=MTRX(5*L-3)
        B2=MTRX(5*L-2)
        B3=MTRX(5*L-1)
        X1=0
        X2=0
        X3=0
        IF(B1.GT.0)X1=C(B1)
        IF(B2.GT.0)X2=C(B2)
        IF(B3.GT.0)X3=C(B3)
        P1=PAR(3*L-2)
        P2=PAR(3*L-1)
        P3=PAR(3*L)
C
C  OUTPUTS OF USER-DEFINED AND UO BLOCKS
C
        A=P1
        B=P2
        YDOT=X1
        F=X2
        C(L+1)=B*YDOT+F
        C(L)=-C(L+1)/A
C
        RETURN
        END
```

APPENDIX H

Iterative Method for Solving an Implicit Equation

The algorithm used to solve an implicit equation $y = f(y)$ is given below (refer Figure 4 and definitions of V and Y blocks in Table 1), where n is the iteration index (i.e. $y(n)$ is the approximation for y after n iterations and $\bar{y}(n)$ is an improved value of $y(n)$):

$n = 0$
$\bar{y}(n) = y_0$ when $t = t_0$
$= y$ at $t - h/2$ when $t > t_0$

(1) $y(n+1) = f(\bar{y}(n))$
If $n > 0$, go to (2)
$\bar{y}(n+1) = y(n-1)$; go to (3)

(2) $a = [y(n+1) - y(n)]/[\bar{y}(n) - \bar{y}(n-1)]$; $q = a/(a-1)$
$\bar{y}(n+1) = q\bar{y}(n) + (1-q)y(n+1)$

(3) If $|\bar{y}(n)| > 10^{-30}$, go to (4)
If $|\bar{y}(n+1)| < 10^{-30}$, go to (5)

(4) If $|\bar{y}(n+1) - \bar{y}(n)| < |\bar{y}(n)|$Ec, go to (5)
$n = n+1$
If $n >$ Ni, print appropriate diagnostic message (see Appendix B) and go to (5)
Go to (1)

(5) $y = \bar{y}(n+1)$; exit

All the quantities calculated above are stored either as block output values or parameters of the blocks used. These quantities are printed at each iteration when using the debugging facility (see Table H.1).

TABLE H.1

Description of V and Y Blocks Corresponding to Debugging Output

Block Type	X	X1	X2	X3	M	P1	P2	P3
V	$\bar{y}(n)$	—	—	—	Nv	y_0	$\bar{y}(n-1)$	$y(n)$
Y	$\bar{y}(n+1)$	$y(n+1)$	$\bar{y}(n+1)$	—	n	Ec	Ni or a*	q

* P2 = Ni for initial and final iterations
$= a$ otherwise

APPENDIX I

(1) Modelling Program Structure: Major Subroutines

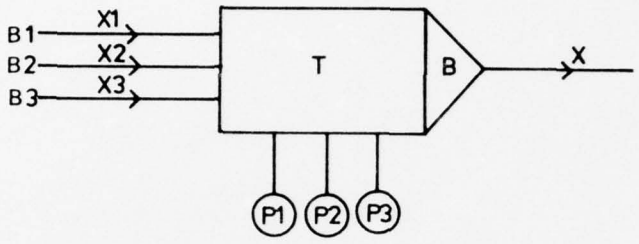
Name(s)	Effect	Section(s) for Reference
MAIN	Controls the flow of the program	—
CSM0	Zeroes appropriate quantities	—
CSM1	Reads in configuration statements	3.1.1, 4.1.4
CSM2, CSM3	Sorts configuration statements	3.2
CSM4	Reads in parameter statements	3.1.2, 4.1.4
CSM5	Reads in function statements	3.1.3, 4.1.4
CSM6	Outputs model specification statements	4.1.5, 5.1.1
CSM7	Reads in integration parameters	4.1.6
CSM8A	Reads in output control parameters	4.1.6
CSM10, CSM11	Executes the model	3.3, 4.1.7
CSM8	*Outputs block values into binary file*	4.2
CSM13	Enables examination of output at final time value	5.5.2

APPENDIX I (continued)

(2) Modelling Program Structure: Service Subroutines and Functions

Name(s)	Effect	Section(s) for Reference
CHARSH*	Used by 'PRNTYP' and 'BLKIN' to shift alpha-numeric characters stored in a FORTRAN variable	—
CHGIFL	When model input is completed for a particular file, used to revert the model input channel to the Teletype	5.1.2
CPU†	Obtains CPU run time	—
DING	Sounds bell on the Teletype	—
EOFCHK	Checks for end of model input file	5.1.2
EXPAND†	Expands an array to the defined number of elements	4.1.2
FCHECK†	Searches for a file on the disk	—
FINPUT, KINPUT	Used to read in data in the form of individual characters	—
GTITLE	Reads in title	4.1.4
INBLKS	Used with 'CSM8A' to read in output block numbers	4.1.6
INOUT, GETNAM	Processes the 'channel' command	5.1
LERROR	Used by dummy versions of user-defined subroutines to record error and type diagnostic error message	4.1.1
OPLBF	Completes block output file	4.2
ORC	Used to determine whether a file may be overwritten	5.2.1
OUT	Outputs debugging information	5.5.3
PACK	Used to pack data into a binary file	4.2
PRNTYP, BLKIN, BLKOUT, CHKTR	Used with 'CSM1' and 'CSM6' to output configuration statements	—
RETAIN	For blocks requiring initial conditions, sets the first parameter equal to the last calculated output value so that execution may be continued at a later stage	5.5.5
RDC	Used to determine whether a file may be read from the disk	5.2.2
RUN*	Allows the user to run another program (e.g. TRANS)	5.5.4
SET, SETCOM	Outputs titling and timing information and block labels	4.2
STORE, FMIN	Used to manipulate function data	3.1.3, 4.1.4
TAGSRC	Searches model input file for a particular type of statement	—
TIMOUT	Outputs CPU run time	—
TTYCHK†	Used to check if "↑" has been typed during a run	4.1.7
UOTEST	Tests for compatibility of UO block with user-defined block	7

* ARL Library subroutines.
† Written in MACRO-10.

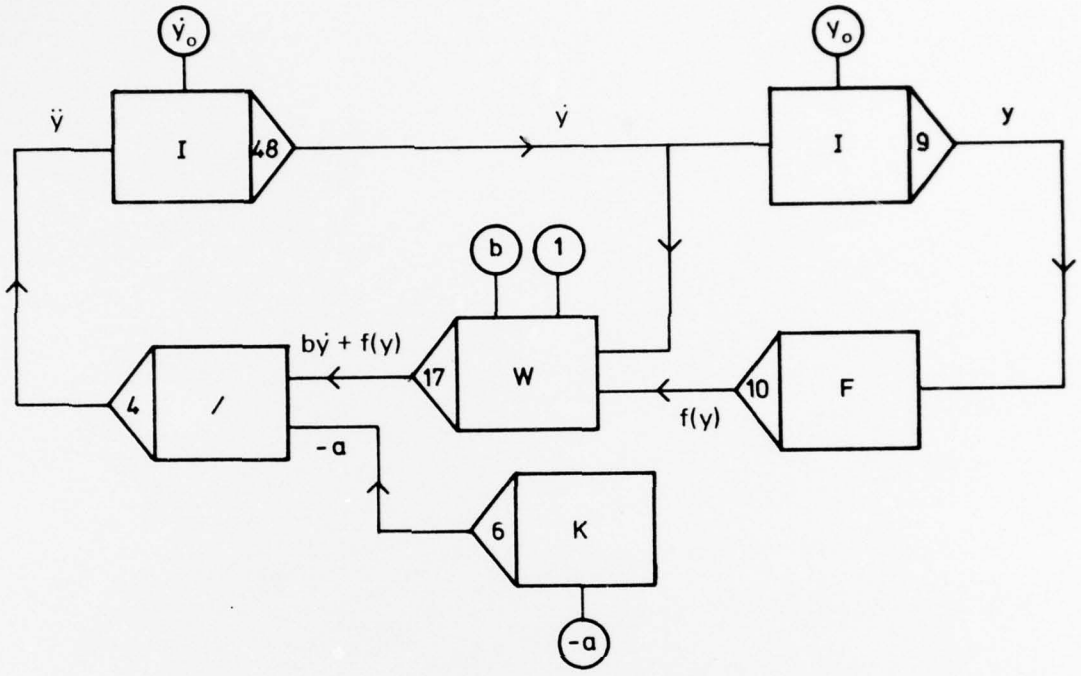


(a) Parameters P1, P2, P3 are shown left to right.
If only one is shown, parameter is P1.
If two are shown, parameters are P1 and P2

(b) Output values X1, X2, X3 of blocks B1, B2, B3 are shown top to bottom.
If only one is shown, it is X1.
If two are shown, they are X1 and X2

BLOCK DIAGRAM NOTATION (refer Section 3.1)

(a) Block diagram (refer Fig. 1)



(b) Differential equation: $\ddot{y} = -[b\dot{y} + f(y)]/a$ (Integration interval, h = 0.1s)

(c) Parameters: $a = 5$, $b = 2$, $y_o = -10$, $\dot{y}_o = 0$

(d) Function f(y):

y	−10	−8	−6	−4	−2	0	2	4	6	8	10
f(y)	−100	−64	−36	−16	−4	0	4	16	36	64	100

(e) Output: y, ẏ, ÿ, f(y) from t = 0 to 20s (Interval = 1s)

(Continued on next page)

(f) Configuration statements

4	/	17	6	$\ddot{y}$
6	K			$-a$
9	I	48		y
17	W	48	10	$b\dot{y} + f(y)$
48	I	4		$\dot{y}$
10	F	9		f(y)

(g) Parameter statements

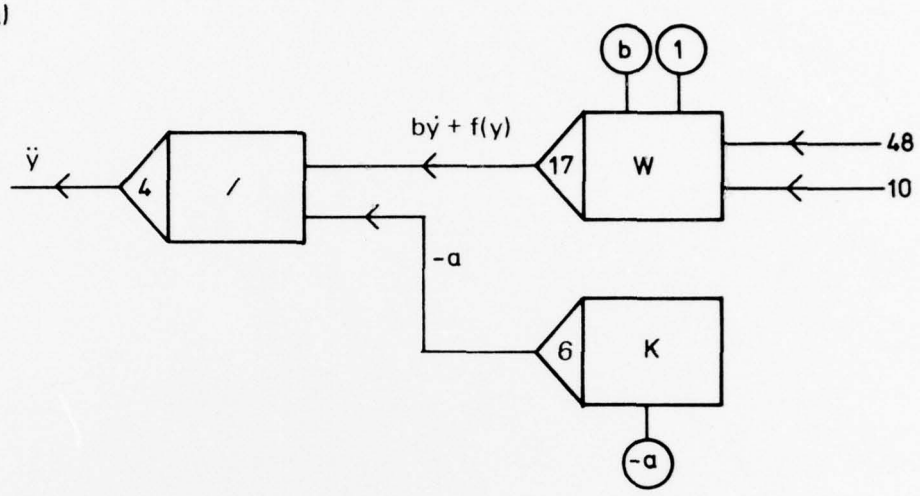
9	−10	
17	2	1
6	−5	

(h) Function statements

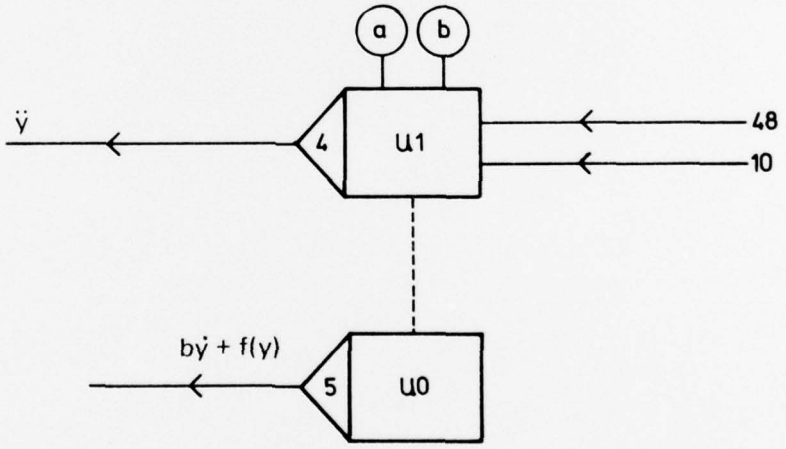
10		block number
−10	−100	coordinate pairs
−8	−64	
−6	−36	
−4	−16	
−2	−4	
0	0	
2	4	
4	16	
6	36	
8	64	
10	100	

NON-LINEAR SPRING PROBLEM

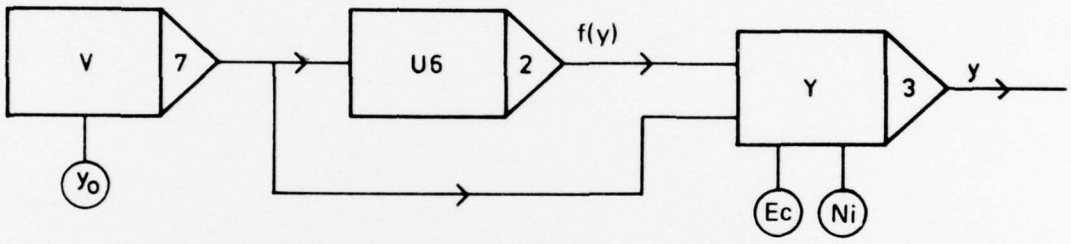
(a)



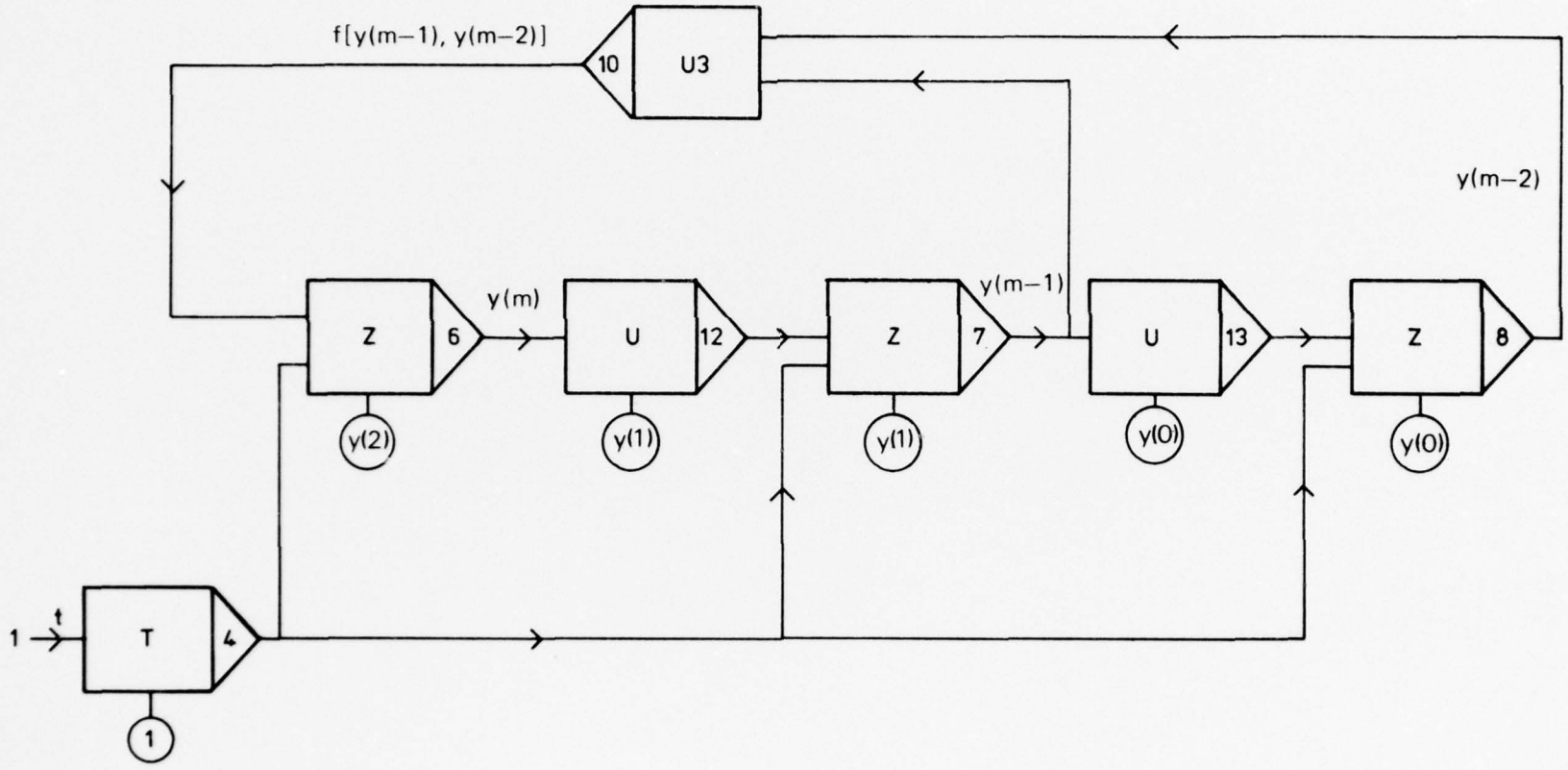
(b)



REPLACEMENT OF (a) DIVIDER, CONSTANT, AND WEIGHTED SUMMER BLOCKS BY (b) USER-DEFINED AND USER OUTPUT BLOCKS



BLOCK DIAGRAM FOR SOLUTION OF THE IMPLICIT EQUATION y = f(y)



BLOCK DIAGRAM FOR A SECOND ORDER DIFFERENCE EQUATION EXAMPLE

AERO NOTE 362
FIG. 5

DISTRIBUTION

Copy No.

AUSTRALIA

DEPARTMENT OF DEFENCE

Central Office

Chief Defence Scientist	1
Executive Controller, A.D.S.S.	2
Superintendent—Defence Science Administration	3
Superintendent—Assistant to Executive Controller, A.D.S.S.	4
Controller—Service Laboratories and Trials Division	5
Controller—Military Studies and Operational Analysis Division	6
Controller—Programme Planning and Policy Division	7
Central Library	8
S.T.I.B.	9
J.I.D.	10

Aeronautical Research Laboratories

Chief Superintendent		11
Superintendent—Aerodynamics Division		12
Aerodynamics Divisional File		13
N. Gilbert	Co-Authors	14–15
P. Nankivell		16–17
L. Mitchell	Structures	18
D. A. Secomb	Aerodynamics	19
D. Hatton	Systems	20
R. Whitten	Systems	21
D. Bird	Mechanical Engineering	22
G. Long	Structures	23
B. W. B. Shaw	Aerodynamics	24
D. C. Collis	Aerophysics Group	25–29
Library		30

Central Studies Establishment

Library	31

Materials Research Laboratories

Library, Victoria	32
Library, New South Wales	33
Library, South Australia	34

Weapons Research Establishment

Library	35
T. J. Packer, Aerospace Division	36
P. Goddard, Systems/Analysis Division	37

Air Office

Army Office

Navy Office

DEPARTMENT OF INDUSTRY AND COMMERCE

Government Aircraft Factory

Victorian Region

Government Engine Works

DEPARTMENT OF TRANSPORT

Air Transport Group

STATUTORY AUTHORITIES AND PRIVATE INDUSTRY

Universities

Bristol	Library, Engineering Department	139
Cambridge	Library, Engineering Department	140
Manchester	Professor Applied Maths.	141
	Professor N. Johannensen	142
Southampton	Library	143

Colleges and Institutes

Cranfield Institute of Technology	Library	144
Imperial College	Library	145

UNITED STATES OF AMERICA

Australian Defence Research and Development Attache	146
Airforce Flight Dynamics Lab., Wright-Patterson A.F.B., Ohio	147
Arnold Eng. Development Center, Arnold Airforce Station, Tennessee	148
National Technical Information Service, Springfield	149
Library of Congress, Gift and Exchange Department, Washington, D.C.	150
Library, National Bureau of Standards, Washington, D.C.	151
N.A.S.A. Scientific and Information Facility, College Park	152–156
Naval Ordnance Labs., Silver Spring, Md.	157
U.S. Army Ballistics Research Labs., Aberdeen Proving Ground, Md.	158
American Institute of Aeronautics and Astronautics	159
Applied Mechanics Reviews	160
Boeing Co., Head Office, Seattle	161
Lockheed Research Labs. (Palo Alto), Sunnyvale, Calif.	162
Pratt & Whitney Aircraft Division, United Aircraft Corporation, East Hartford	163

Universities

Cornell (New York)	Library, Aero. Labs.	164
Carnegie–Mellon (Pennsylvania)	Library	165
Johns Hopkins	Library	166

Institutes of Technology

California	Library, Guggenheim Aero. Labs.	167

Spares 168–192

DOCUMENT CONTROL DATA

1. Security Grading/Release Limitation

   (a) Document Content: Unclassified

   (b) This Page: Unclassified

2. Document Type/Number
   Aerodynamics Technical Note
   362

3. Document Date
   May 1976

4. Title and Sub-Title THE SIMULATION LANGUAGE CSMP-10(ARL)

5. Personal Author N. E. GILBERT and P. G. NANKIVELL

6. Corporate Author(s) AERONAUTICAL RESEARCH LABORATORIES

7. *ABSTRACT*

A description is given of the 'block oriented' simulation language CSMP-10(ARL), which has been developed from CSMP-10 and is written mainly in FORTRAN IV for a PDP-10 computer. The major improvement made has been to incorporate 'user-defined' blocks, which are written as FORTRAN subroutines. A large number of outputs may be defined within these subroutines by using 'dummy' or 'user output' blocks. Two other major improvements have been made, both enabling the saving of appreciable core storage. Firstly, arrays necessary to store information on each block used are automatically expanded to the size determined by the user. Secondly, the language is divided into a modelling program, which is used to perform the model simulation and store the output in a binary file on a specified storage device, and an output program, which is used to print and plot the character conversion of the binary file.

8. Computer Program(s) — Titles and Language
   BOMMP
   TRANS (FORTRAN IV)

9. Descriptors
   Simulation Languages
   Simulation
   Mathematical Models
   Computerized Simulation

11. Cosati Classifications
    0902, 1407, 1201, 1402

12. Task Reference (RD/P)
    RD74/22

10. Library Distribution (Defence Group)
    Central Office Bridges Library
    S.T.I.B.
    J.I.O.
    A.R.L.
    C.S.E.
    M.R.L., Vic., N.S.W., S. Aust.
    W.R.E.
    A.M.T.S.
    A.R.D.U.
    E.D.E.

13. Sponsoring Agency Reference
    A.R.L. File A2/19

14. Cost Code
    51 7740

15. Imprint MELBOURNE—AERONAUTICAL RESEARCH LABORATORIES, 1976